An Ada object oriented missile flight simulation.

Waite, John V.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN ADA OBJECT ORIENTED
MISSILE FLIGHT SIMULATION

by

John V. Waite

September 1991

Thesis Advisor: Yuh-jeng Lee

Approved for public release; distribution unlimited

UNCLASSIFIED
ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b. OFFICE SYMBOL (If applicable) Code CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
An Ada Object Oriented Missile Flight Simulation

12. PERSONAL AUTHOR(S) John V. Waite

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM ________ TO ________	September 1991	195

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Ada, Software Engineering, Object Oriented, Simulation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis uses the Ada programming language in the design and development of an air-to-air missile flight simulation with object oriented techniques and sound software engineering principles. The simulation is designed to be more understandable, modifiable, efficient and reliable than earlier FORTRAN simulations. The principles of abstraction, information hiding, modularity, high cohesion and low coupling are used to achieve these goals. The resulting simulation is an accurate mapping of the problem space into software. The simulation is a three Degree-of-Freedom (3-DOF) model of RF/IR guided air-to-air missile. Two targets are also modeled. The simulation is primarily intended to study missile kinematics.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Professor Yuh-jeng Lee	22b. TELEPHONE (Include Area Code) (408) 646-2361	22c. OFFICE SYMBOL Code.CS

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

AN ADA OBJECT ORIENTED MISSILE FLIGHT SIMULATION

by

John V. Waite
B.S., Wayne State University, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

ABSTRACT

This thesis uses the Ada programming language in the design and development of an air-to-air missile flight simulation with object oriented techniques and sound software engineering principles. The simulation is designed to be more understandable, modifiable, efficient and reliable than earlier FORTRAN simulations. The principles of abstraction, information hiding, modularity, high cohesion and low coupling are used to achieve these goals. The resulting simulation is an accurate mapping of the problem space into software. The simulation is a three Degree-of-Freedom (3-DOF) model of RF/IR guided air-to-air missile. Two targets are also modeled. The simulation is primarily intended to study missile kinematics.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

The ever increasing cost and complexity of modern weapon systems forces new demands on the test and evaluation (T&E) process. More extensive testing is required with fewer resources. This thesis explores one aspect of the T&E process as it relates to air-to-air guided missiles.

In the early days of missile T&E (*circa* late 1940s), missile performance capability was determined solely through flight test, that is, actual missile launches. The realization that all the T&E data requirements could not be met with a limited number of launches led to captive-carry flight test, laboratory testing, and simulation to complement the missile launches. Today's data requirements have grown in response to the increased missile sophistication and mission complexity. It is not unusual for a single flight test to cost more than a million dollars. Due to the increased data requirements and increased cost of flight test, missile flight simulation is receiving more and more attention.

There are three levels of missile flight simulation in terms of cost and complexity. Real-time hardware-in-the-loop (HIL) simulation integrates actual missile hardware with special test and instrumentation equipment in a laboratory environment. The simulation software typically runs on a high-speed special purpose computer that drives the test equipment and missile hardware. The real-time HIL simulation requires a major development effort of approximately thirty-five to forty man years and costs from five to ten million dollars [Ref. 1]. The second level of simulation is the all digital six-degree-of-freedom (6-DOF) missile flight simulation. Six-degree-of-freedom indicates that the simulation computes forces and moments for all three axes. The 6-DOF simulation incorporates sophisticated models for various missile subsystems and runs on a mainframe class computer. The 6-DOF runs many times slower than real-time. For example, an actual missile flight that might take thirty seconds to complete in real-time might take eighteen hours to run to completion using a 6-DOF simulation. The 6-DOF simulation requires a development effort of 4-6 man years.

This thesis will concentrate on the third level of simulation, the fast analytical simulation (FAS). Simulations of this class are a rapid and inexpensive tool allowing missile systems analysts to study overall missile response or capability expeditiously. The FAS is a three degree-of-freedom (3-DOF) simulation, usually the forces are computed for all three axes (the moments are ignored) and a three dimensional space is represented. Alternately, a 3-DOF might represent a planar two dimensional view where forces acting on two axes are computed while moments are computed about the remaining axis. The FAS is intended to be easily accessible via personal computers to provide results in a timely fashion. A user enters initial conditions and results are presented within a few minutes.

B. CURRENT PRACTICES: PROBLEMS AND LIMITATIONS

The same problems are common to all three levels of missile flight simulation. The basic problems are that the simulations are extremely difficult to understand and to modify. The causes of these problems stem from the methods (or more accurately the lack of methods) and language used to implement the simulations. The difficulty in understanding and modifying the simulations introduces problems with efficiency and reliability.

The simulations are usually developed by physicists or aerospace engineers using the FORTRAN programming language. Their main goal is "just to get something up and running". Typically these people have little or no training in modern software engineering principles. The resulting simulations are poorly structured and violate most commonly accepted programming principles. Typical characteristics of these simulations are:

1. The simulations are monolithic pieces of code using many GOTO statements.
2. Most variables are treated as global.
3. Common data areas are used for communication between subroutines.
4. Cryptic variable names are used (FORTRAN variable names are limited to six characters).
5. The simulations are limited to very simple data structures (multi-dimensional arrays are usually the most sophisticated data structures found).
6. Programming through side is common.
7. The simulations have little or no comments or formal documentation.

The new analyst will usually require at least six months to gain a basic understanding of how the simulation works, even if he or she has an excellent understanding of missile systems. An understanding of the simulation is critical if results are to be interpreted correctly. It is not uncommon for the original developers of a simulation to move on to other jobs, leaving the analysts responsible for maintaining and modifying the simulation. Changes in the production missile's software or hardware, regularly occurring events, must be accurately reflected in the simulation code. Changes or patches introduced to the simulation code invariably make the code more obscure and, more often than not, produce undesirable side affects or bugs. Debugging these types of problems is incredibly time consuming and difficult.

After numerous patches have been applied the simulation software becomes unreliable and inefficient. Wildly different results are obtained for slightly different initial conditions. The real-time HIL simulations no longer run in real-time. The 6-DOF simulations may take days to solve a problem and the 3-DOF FAS simulations take hours - what once required hours and minutes respectively. Disk and main-memory capacity become issues. What was once a tool enabling scientists and engineers to analyze complex systems has become an unwieldy demanding burden of questionable value.

C. MOTIVATION AND GOALS

Current missile flight simulations are difficult to understand and modify, hence inevitably become inefficient and unreliable. What is needed is a method that more closely represents the problem space allowing simulations to be developed that are easy to understand and modify. This thesis will explore the use of object oriented techniques using the Ada programming language, in conjunction with contemporary software engineering principles, to implement a missile flight simulation. This simulation should be easily understood in a reasonable amount of time and readily accommodate change. Additional goals include producing code that is efficient and reliable.

D. THESIS ORGANIZATION

Chapter II presents a brief view of object oriented development philosophy and technique using Ada. Chapter III discusses the problem space, the flight simulation of an air-to-air guided missile. Chapter IV presents a user-level view of the simulation. Chapter V concentrates on the simulation control and support objects, while Chapter VI discusses the missile, launcher, and target objects. Thesis conclusions and recommendations are presented in Chapter VII.

II. OBJECT ORIENTED TECHNIQUES WITH ADA

A. OBJECTIVES OF OBJECT ORIENTED TECHNIQUES

The goal of object oriented techniques is to produce software that is understandable, easy to modify, efficient, reliable and reusability. Object oriented techniques build on sound software engineering principles to encapsulate data and procedures into objects. Object oriented techniques, which capture the real world problem space, map well into the Ada programming language.

1. Understandability

Understandability is critical to the management of complex software systems. It is, without a doubt, the most important factor of a simulation to an analyst or person responsible for maintaining the simulation. The software solution, that is the simulation, should be an accurate model of the real world problem. Software can be thought of as being understandable on both a micro and macro level. Code at the micro level should have a style that is very readable. At the macro level data structures and algorithms should be able to be identified as mapping from the real world problem space. Understandability also tends to be tied to the programming language used and its richness of expression.

2. Modifiability

Well designed software should readily permit change. Modification is usually required due to a change in requirements or to correct to an error. Changes in missile simulation code are required to explore new concepts, or as a result of missile hardware or software upgrades. Many changes are not planned. Ideally changes should not alter the fundamental architecture of the software solution.

3. Efficiency

Efficiency is the optimal use of two fundamental computer resources - storage space and execution time. Both of these resources are dependent underlying hardware, yet both resources are equally dependent on the software. An efficient missile simulation should provide better user response and more functionality than an inefficient simulation.

4. Reliability

The goal of reliability is to prevent failure, and to some extent, recover from failure in a graceful manner. Failure in a missile simulation might be defined as anything from a program that crashes to a program that produces results that do not agree with flight test data or produces inconsistent results. A reliable missile simulation will provide results that are consistent with real world experiences and give meaningful indications when potential problems might arise (e.g., limits exceeded or incorrect user input).

5. Reusability

The goal of reusability is to provide software components to build software much the same way hardware engineers build circuits from standard off-the-shelf components. The development of software systems can be dramatically reduced by using software components that have already been debugged and tested. These components can form libraries of commonly used objects. Systems may be constructed from these libraries. These systems then may be added to the library.

B. OBJECT ORIENTED PRINCIPLES

Through abstraction, information hiding and modularity, object oriented techniques encapsulates data and procedural abstractions to form objects. Objects modularize both information and processing, rather than processing alone. Object oriented techniques establish a mechanism for (1) a representation of data structures, (2) the specification of process and (3) the invocation procedure. An object is an element of the real world mapped into the software domain. The object consists of operations which act on data structures in response to messages sent to that object from other objects. The operations and data structures are hidden, that is the implementation details are unknown to the user of the object. The interface to the object is the only portion visible to the user. The interface is a set of well defined messages that specify what operation on the object is desired. Object oriented techniques can aid sound software engineering principles. These principles include abstraction, information hiding, modularity, loose coupling, and strong cohesion [Ref. 2].

1. Abstraction

Many of the problems found with the missile flight simulations are due to their complexity. Abstraction is a powerful concept that helps one deal with complexity. Abstraction concentrates on the essential aspects of a problem, while omitting the details. There may be many levels of abstraction constructed when solving a problem. At the top level of a missile simulation, abstraction would reveal the essential entities - the missile, target, and environment. Moving to the next lower level of abstraction within the missile, this level might be thought of as being composed of various subsystems, such as the seeker, the guidance section, the autopilot, and the airframe. Moving to the next lower level of abstraction, arbitrarily choosing the missile seeker for example, would reveal the data structures and procedures used to model the seeker. Only the lower levels of abstraction expose the specific details of a solution.

2. Information Hiding

Information hiding conceals the implementation details of a solution that should not affect other parts of a system. Through information hiding only the essential aspects of a solution are visible, while the implementation details or "how" of a solution are hidden. Hiding low level design decisions prevents the higher levels of abstraction from being dependent on implementation details. This approach aids abstraction and increase the modifiability of the solution.

3. Modularity

The importance of modularity in software design has been recognized for some time. According to Myers [Ref. 3], "Modularity is the single attribute of software that allows a program to be intellectually manageable." In monolithic software, such as the missile simulations, the number of control paths, number of variables and the overall complexity make understanding difficult. Ideally, software is decomposed into modules along logically and functionally independent lines. Modularity supports our notion of abstraction. High-level modules specify "what" is to be done. Low-level modules specify "how" that action is to take place.

3. Cohesion and Coupling

Modules in software systems can be thought of as having two important characteristics, cohesion and coupling. Cohesion attempts to characterize to what degree a module performs a single function or serves a single purpose. A highly cohesive module would be one in which the module performs a single task that requires little or no interaction with other modules in a program. A module exhibiting low cohesion would perform many different functions and interact with a large number of other modules. Modules that are highly cohesive are easier to understand and are more amenable to change than modules exhibiting low cohesion.

Coupling is measure of interconnection among modules in a program. Modules with high coupling have a complex interfaces and make use of data or control information found in other modules. Modules with low coupling have relatively simple interfaces and make use of only the data or control information presented by the interfaces of other modules. Changes made to modules with low coupling are less likely to cause unwanted effects in other modules, that is the ripple effect is minimized. Like modules that are highly cohesive, modules with low coupling are easier to understand and modify.

4. Inheritance

Inheritance is an object oriented concept that permits the organization, building and reuse of software [Ref. 4]. In a limited view of this concept, new objects may be defined to inherit the capability and functionality of other previously defined objects. The new objects may extend the capability and functionality of the original object by adding new capabilities and functionalities. Conversely the new object may be defined to eliminate or limit certain capabilities of the original object. Once an object has been developed, it may be reused with minimal effort through inheritance, reducing development time.

C. OBJECT ORIENTED METHODOLOGY WITH ADA

1. Ada Packages

The object oriented philosophy maps well into the Ada programming language. Ada has a wide set of constructs for providing primitive objects and operations. These constructs serve to build the implementation level of the objects. Ada's packaging concept is conceptually similar to objects and provides the means to encapsulate objects. According to Booch [Ref. 5], "A package is a collection of computational resources, which may encapsulate data types, data objects, subprograms, tasks or even other packages." An Ada package consists of a specification and a body [Ref. 6]. The specification identifies the information that is visible to the user of that package. The package body contains the implementation details of the package which should (and can) remain physically and logically hidden from the user. The specification and body may be compiled separately to enforce the separation of the specification or interface from the body with its implementation details. The specification can serve to define the messages associated with an object. The object responds in the appropriate manner to these messages. These messages might map to function or procedure calls and their input or output variables.

Ada packages can be used to provide reusable software components. Packages of commonly required objects can form libraries where they may be withdrawn and reused. Ada's generic unit feature supports, in a limited way, the object oriented principle of inheritance. A generic package serves as a template for an object [Ref. 7]. The generic object can then be instantiated with all the features of the generic object, along with any additional features required of that particular instantiation. For example, a generic stack or list object might be instantiated for each occurrence of a different data type, along with the additional capabilities that make sense for that particular data type.

2. Methodology

This thesis uses an object-oriented development technique similar to that advocated by Booch [Ref. 5] and first proposed by Abbott [Ref. 8]. The development process involves five steps. First, identify the objects and their characteristics or attributes as they exist in the problem space. Often a concise problem statement is useful in identifying objects. The nouns of the problem statement serve to identify potential objects. The second step is to identify the operations that characterize the behavior of the objects identified in the first step. These should be meaningful operations that can be performed on the object. Verbs associated with an object noun in the problem statement can aid in the identification of meaningful operations. During this step time and space constraints are formed to define the dynamic behavior of the objects. The scope and ordering of operations might be defined for example. The third step is to establish the visibility of the objects with relation to one another. This step attempts to specify what objects "see", and what are "seen" by a given object. This serves to map the problem space into the objects. The fourth step is to define the interfaces to the objects. To do this an object specification is produced which "forms the boundary between the outside view and the inside view of an object." This maps directly into the Ada package specification construct. The final step is to implement each object by designing suitable data structures and algorithms and to implement the corresponding interface from the fourth step. Also at this step it is important to remain aware of the software engineering principles of modularity, high cohesion and low coupling. Note that this whole process can be recursive, that is, an object might further be decomposed into subordinate objects.

The key point of this method is the accurate mapping of the problem space into software. This mapping preserves the real world view of the problem, and if done properly, tends to produce code that is easily understood. Object oriented techniques also lend themselves well to the software engineering principles discussed earlier. Through object oriented techniques and sound software engineering principles, our goals of producing a missile flight simulation that is easy to understand, easy to modify, efficient and reliable can be realized.

III. THE PROBLEM SPACE

A. INTRODUCTION

An air-to-air guided missile is designed to be carried on an aircraft and launched at an airborne target. After launch the missile guides, using its sensors, on the target to intercept. Air-to-air missile sensors may be radar, infrared or a combination of both types. Once the missile detects the target, it tracks and guides on the target by generating steering commands that will set a course to intercept the target. The missile flight simulation attempts to represent or model the missile and its environment.

The missile flight simulation models a subset of the missile systems, the kinematics - consisting of the missile dynamics and the missile-target geometry, and the target. At the top level view, the simulation computes the forces acting on the missile (e.g., thrust, drag, and gravity) and from these forces derives accelerations to compute the missile's spatial trajectory from launch to target intercept. Figure 3.1 is a top-level block diagram of a missile flight simulation indicating the relationships of the various models. The missile subsystems are represented by the Autopilot, Airframe, and Guidance blocks. The blocks labeled Missile Dynamics and Missile-Target Geometry compute the kinematics. The Target block here represents a single target, but in most simulations more than one target is modeled. The missile airframe model, given its achieved accelerations, computes the forces acting on it for use in the

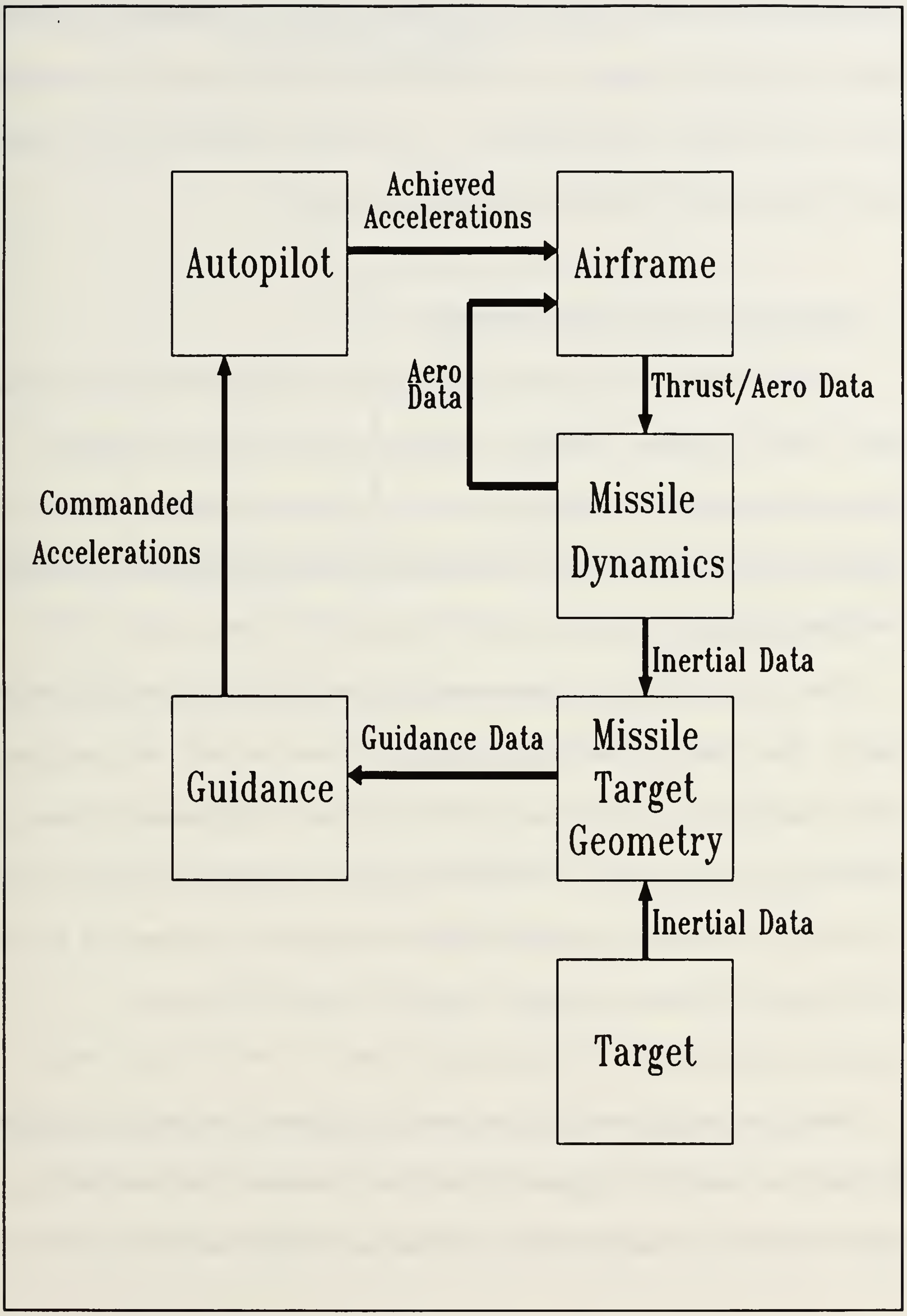


Figure 3.1 Missile Flight Simulation

target geometry, which is passed to the guidance model. Acceleration commands, which will enable the missile to intercept the target, are computed by the guidance model and provided to the autopilot. The autopilot responds with the achieved accelerations, which are passed to the airframe model.

B. THE AIR-TO-AIR GUIDED MISSILE

The missile consists of a number of subsystems. These typically are the airframe, flight control, guidance, warhead, propulsion, data link and telemetry. The subsystems modeled in a simulation are the airframe, autopilot, guidance, and propulsion [Ref. 1]. In addition, the atmosphere and kinematics are modeled.

1. The Airframe

The missile airframe actuates or deflects the control surfaces which steer the missile. The missile is modeled as a rigid body and, as such, body and control surface bendings are not represented. The airframe is represented in terms of a reference axes system. Figure 3.2 illustrates the missile reference axes systems where x, y, and z are the primary missile reference or body axis system and the axes with the a subscript represent the missile autopilot or inertial axis system.

Roll (ϕ) is angular motion about the x-axis, pitch (θ) is angular motion about the y-axis, and yaw (ψ) is angular motion about the z-axis. The missile's angle of attack, alpha (α), is the angle between the missile's velocity vector (V) and its x-axis.

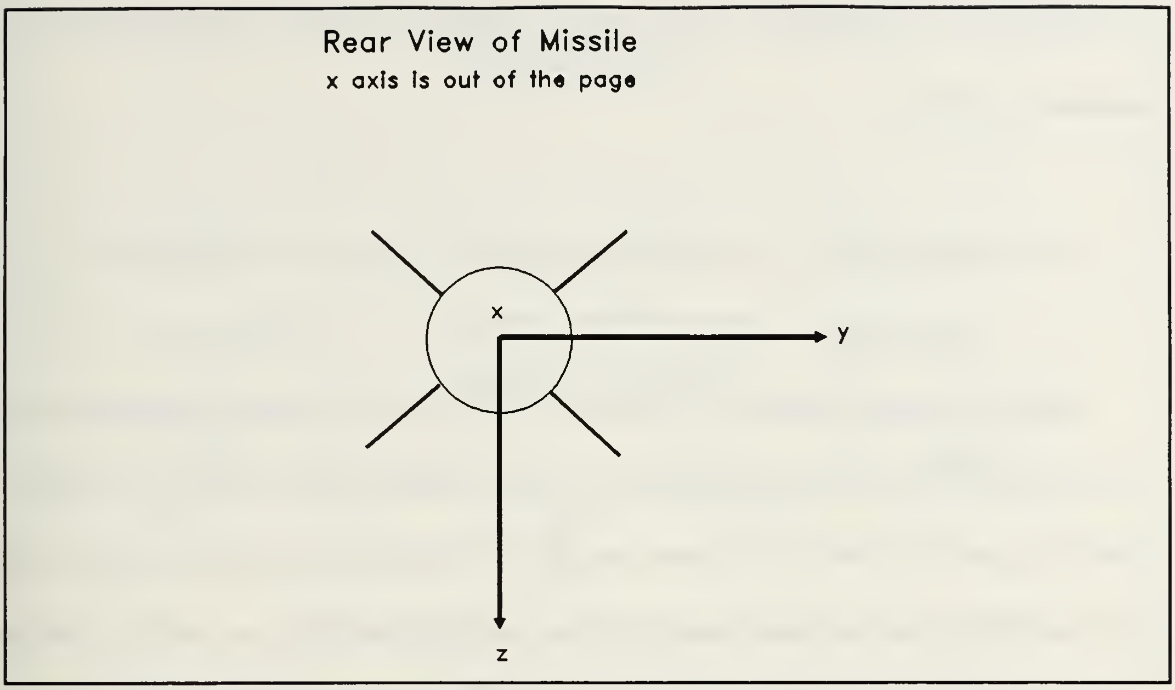


Figure 3.2 Missile Reference Axes

Three force equations describe the forces experienced by the missile along each axis. There is one force equation for each axis as follows:

$$\Sigma F_x = m * a_x$$

$$\Sigma F_y = m * a_y$$

$$\Sigma F_z = m * a_z$$

These represent Newton's classic relation that force is the product of mass and acceleration. Here we resolve the forces into components along each missile axis.

These force equations describe the dynamics of the airframe. Aerodynamics is the science applied to predicting these forces. These forces are expanded in terms of aerodynamic parameters and coefficients [Ref. 9]. For example, the x-axis equation becomes:

$$\Sigma F_x = m * a_x = Fp + (Cd_\alpha * \alpha q * S) + (Cd_\delta * \delta * q * S)$$

The first term in the above equation is the propulsion force. The second term is the product of the drag coefficient for a given angle of attack (Cd_α), the angle of attack (α), and the missile reference area (S). The third term is the product of the drag coefficient for a given control surface deflection (Cd_δ), the control surface deflection (δ), the aerodynamic pressure (q), and the missile reference area (S). The aerodynamic coefficients are a function of angle of attack, control surface deflection, roll angle and mach number. The aerodynamic parameters and forces are provided to the airframe model by the missile dynamics model, and the autopilot model provides the commanded accelerations as input. The airframe model computes the forces it is "experiencing" and sends these values to the kinematics model (see Figure 3.3).

2. Kinematics: Missile Dynamics

The kinematics model serves two functions: to compute missile dynamics and to compute the missile-target geometry (see Figures 3.4 and 3.5). Inputs to the missile dynamics function are the airframe forces computed in the airframe model. From these values, and from initial conditions, the dynamics model derives

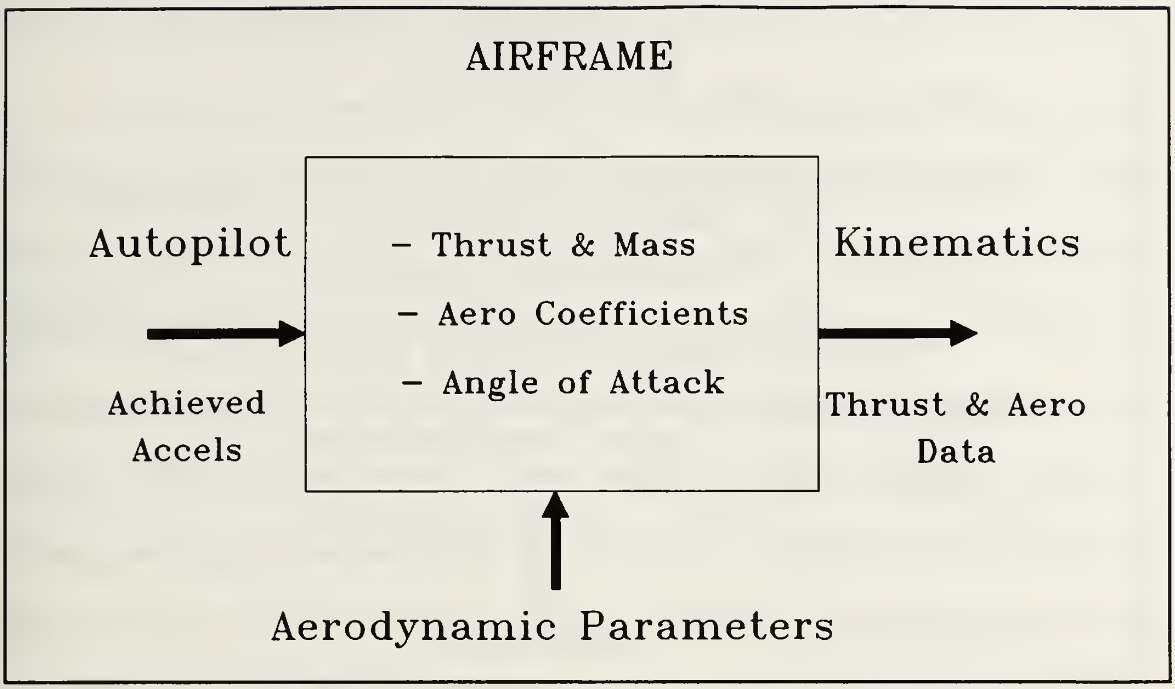


Figure 3.3 Airframe Model

acceleration, velocity, position data, and flight-path variables. Angles, angular rates, and accelerations represent the inertial quantities. These inertial quantities simulate the inertial sensor measurements the missile would experience.

Aerodynamic parameters, such as mach and velocity, are fed back to the airframe model. The derived acceleration, velocity, and position variables are sent to the missile-target geometry model. The kinematics model also transforms data between the two reference coordinate systems, that is, between the airframe reference system known as the body coordinate system (x,y,z) and the autopilot reference system known as the inertial coordinate system (Xa,Ya,Za).

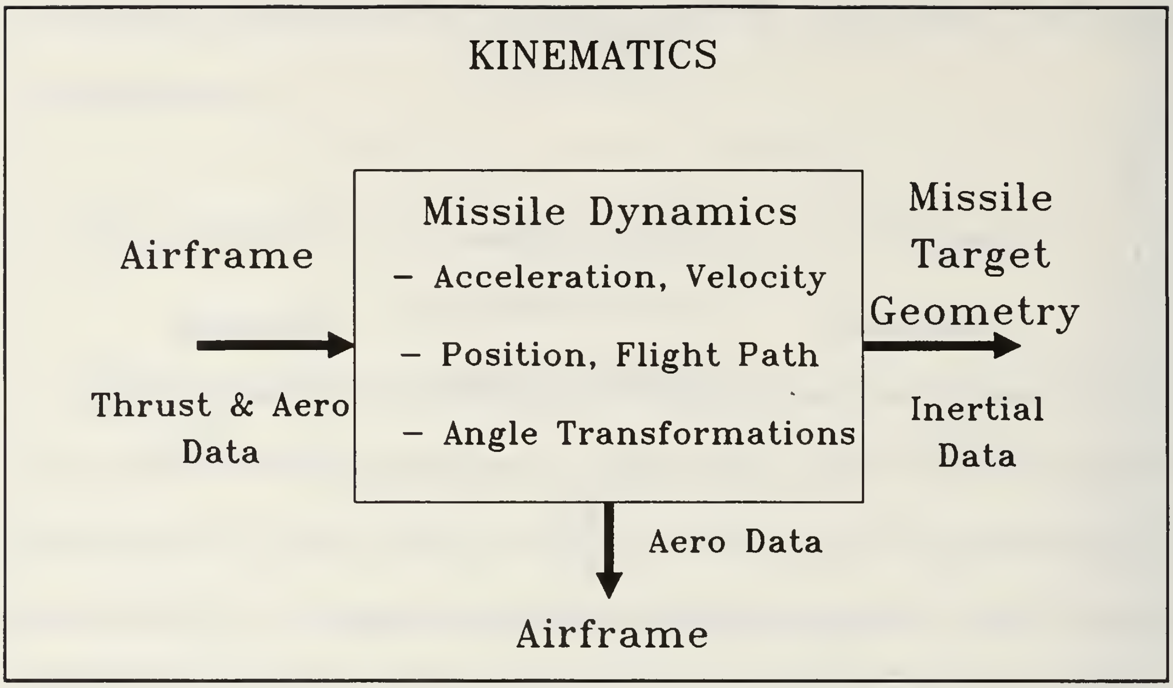


Figure 3.4 Kinematics: Missile Dynamics

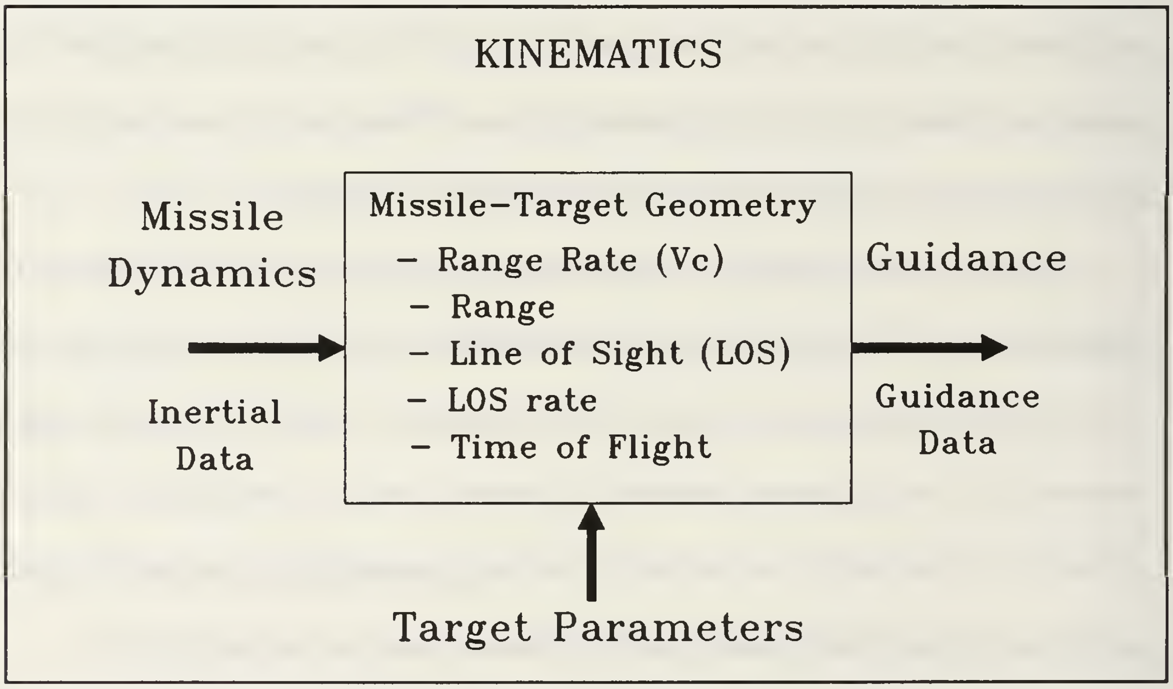


Figure 3.5 Kinematics: Missile-Target Geometry

3. Kinematics: Missile-Target Geometry

The primary purpose of the missile-target geometry portion of the kinematics model is to compute the missile-target engagement geometry parameters. These values are sent to the missile guidance model to steer the missile to the target. Inputs to the missile-target geometry model are missile acceleration, velocity, and position data from the missile dynamics model, and target acceleration, velocity, and position from the target model. These inputs are used to compute range rate (closing velocity), missile to target range, line-of-sight (LOS) rate, LOS, and time of flight. The simulation is terminated on range or time constraints determined by this model. The computed information is sent to the missile guidance model.

4. Missile Guidance

The guidance model represents the missile guidance law (see Figure 3.6). The guidance law determines what trajectory will cause the missile to intercept the target. Missile guidance can be classified by the type of sensor is used to provide target information. Common sensors are RF (radar), or infrared (IR). A missile may use a combination of RF and IR seekers. The actual missile guidance section is very complex and sophisticated, hence, extremely difficult to model. Most simple simulations assume a perfect guidance section that uses a modified proportional guidance law.

An important parameter in guidance is the line-of-sight (LOS). The line-of-sight is the direction the missile "looks" in order to "see" the target. This is an

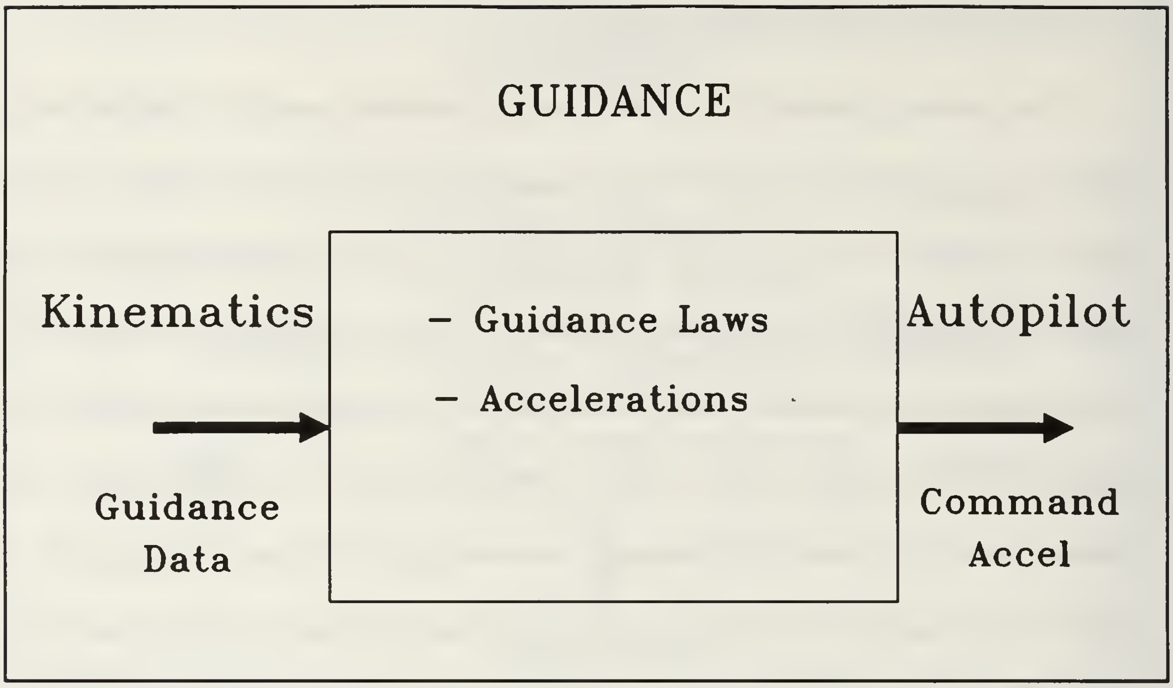


Figure 3.6 Guidance Model

imaginary line from the missile's seeker to the centroid of the target. It has been proven that, given constant target and missile velocities, if the LOS angle between the target and the missile remains constant an optimum trajectory will be achieved, resulting in a minimum miss distance [Ref 1]. If the LOS angle is to remain constant then the LOS rate must be zero. The LOS rate is computed in the guidance section and multiplied by the navigation ratio N and sent to the autopilot as commanded accelerations proportional to the LOS rate; hence, the term proportional guidance.

The commanded accelerations will change the missile velocity relative to the target velocity, driving the LOS rate to zero. The response of the guidance system is determined by the value chosen for the navigation ratio N. Most modern missiles improve upon the pure proportional guidance law by using the target related information available through improved sensors and increased on board computing power. Target related parameters used in addition to LOS include target range, velocity, acceleration and time to intercept. Given guidance data, the guidance model computes the commanded accelerations required to intercept the target.

5. The Autopilot

The autopilot functions to give the missile stable and controlled flight. The autopilot has its own axes reference system (Xa, Ya, Za). The airframe motions about the autopilot axes are controlled by the autopilot. Motions about the Ya and Xa axes determine missile direction. The autopilots for these axes are termed the pitch and yaw autopilots, respectively.

The autopilot receives commanded accelerations as input and responds with achieved accelerations as output (see Figure 3.7). The achieved accelerations are based on the characteristics of the autopilot and other missile subsystems.

6. Target

The target model represents a simple maneuvering target. Inputs are positional, heading, velocity, and type of maneuver initial conditions. Inertial data are derived from this data and output to the missile-target geometry model. More sophisticated target models might include multiple targets and targets capable of complex maneuvers.

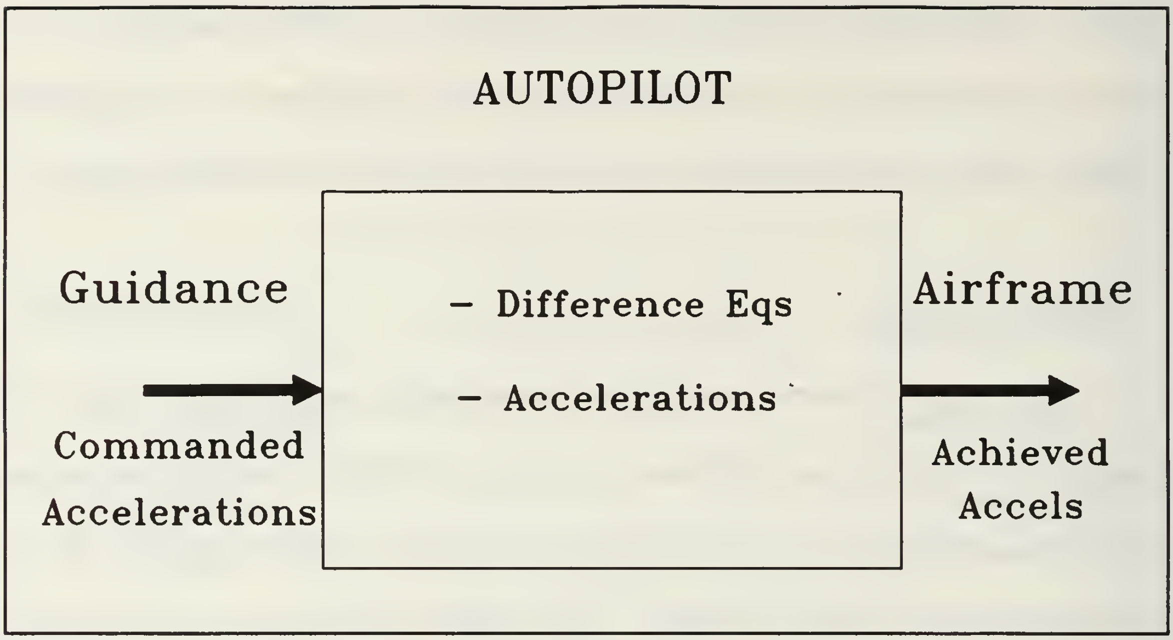


Figure 3.7 Autopilot Model

7. The Atmosphere

The Earth's atmosphere is a dynamically changing system, within which the missile must operate. The pressure, density and temperature of the atmosphere depend on altitude, location on the globe, the time of day and the season. In order to have a common reference atmosphere, a standard atmosphere has been defined by the U.S. Air Force [Ref. 9]. The standard atmosphere gives mean values of pressure, density, and temperature as a function of altitude. Most missile flight simulations model the standard atmosphere.

C. SUMMARY

The models described above become the Ada objects in the object oriented approach. The relations between the models are represented by messages between the objects. These messages can request actions of objects or be in response to message requests for action. Objects may be constructed from other objects. A missile composed of objects that represent subsystems for example. This approach results in a simulation that accurately maps the problem space to software, as the next chapter illustrates.

CHAPTER IV. THE USER'S VIEW OF THE SIMULATION

A. INTRODUCTION

This chapter briefly discusses general simulation principles, and contains a basic user-level overview of the simulation. The brief discussion of simulation principles is intended to provide a rudimentary understanding and insight into some aspects of the simulation's implementation and operation. The user-level overview of the missile flight simulation operation provides basic concepts that will aid in understanding the missile, launcher, and targets objects presented in Chapter VI.

This missile flight simulation falls under the category of nonlinear continuous dynamic systems [Ref. 10]. Other simulations or applications in this category include simulations of spring-damper systems, automotive drive trains, power plants, and chemical processing plants. In keeping with the principle of abstraction, the top-level of the simulation is generic in the sense that it is not designed for a specific simulation, it is application independent. The application could be any of the previously mentioned simulations. The principle common to all these simulations is that the application specific models (e.g., a power plant or a missile) produce a set variables that represent the "state" of the model. Typically the state variables are time-dependent variables that represent rates, such as flow rates in a power plant or accelerations in a missile. The specific application models the system of interest over a period of time. This might be hours or days for a power plant, or seconds or minutes for a missile. The time periods are divided into discrete units of time called time steps. A time step for a power plant might be an hour, while a missile's time step might be a tenth of a second. The state variables are computed once each time step by the models. These state variables are then mathematically integrated each time step using numerical integration methods [Ref. 11]. The integrated variables are then fed-back to the models where they are used, along with other equations representing characteristics of the model, to compute the state variables for the next time step [Ref. 12].

B. SIMULATION OPERATION

The simulation is a 3 Degree-of-Freedom (3-DOF) simulation representing translational motion in three dimensional space. The missile is a dual mode, guided air-to-air missile. Dual mode indicates that the missile utilizes both radar and infrared sensors (also known as seekers) to guide to or track the target. The missile is carried on a launch aircraft and is launched at the target. The function of the missile is to guide to and intercept one of the two targets. The targets may fly a constant nonmaneuvering flight path or the user may enable target maneuvering. The targets are capable of both turn and weave maneuvers. In addition to the two targets, it is possible to have two other targets active which act as stand off jammers (SOJs). The role of the SOJs is to use electronic counter measures (ECM) to degrade or confuse the missile's radar to prevent the missile from guiding on the target.

An inertial coordinate system is used to describe the launch aircraft, missile, target and SOJ positions in three dimensional space. This is a north, east, down coordinate system (north, east, down) that is referenced to the missile's position at launch (i.e., (0, 0, -launch aircraft altitude)). An object's position and velocity in inertial space are described by its state vector. In addition to the inertial coordinate system, the missile uses the missile body coordinate system and the seeker coordinate system. The missile body coordinate system is referenced to the airframe or body of the missile and is used to compute forces acting on the missile. The seeker coordinate system is referenced to the missile's seeker and is used to describe where the seeker is pointing with respect to the missile body and the other objects in inertial space.

Sample planar views of a typical launch profile are provided in Figure 4.1, a top view, and Figure 4.2, a side view. All the parameters required to establish a default launch profile are read from a data file when the simulation is first brought up. The user enters SIM at the DOS prompt to bring up the simulation. An introduction screen is displayed, and any key stroke will then display the Simulation Main Menu (see Figure 4.3).

This menu's choices are: File Operations Menu, Launch Aircraft Parameter Menu, Target Parameter Menu, Start Simulation or Quit Program. The File Operations Menu, shown in Figure 4.4, permits the loading and saving of data files that establish the simulation's initial conditions. This menu also has a selection that allows the data generated by the simulation to be logged to a file and the selection of the log

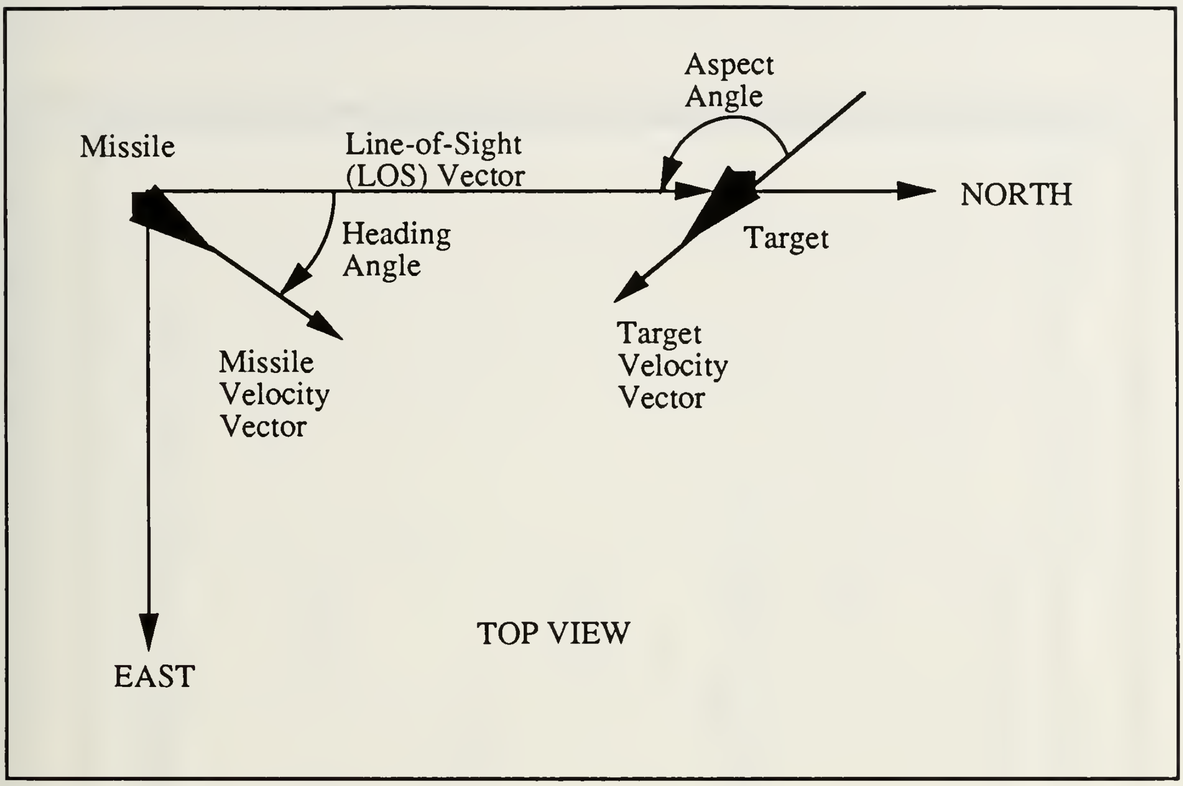


Figure 4.1 North-East Planar View

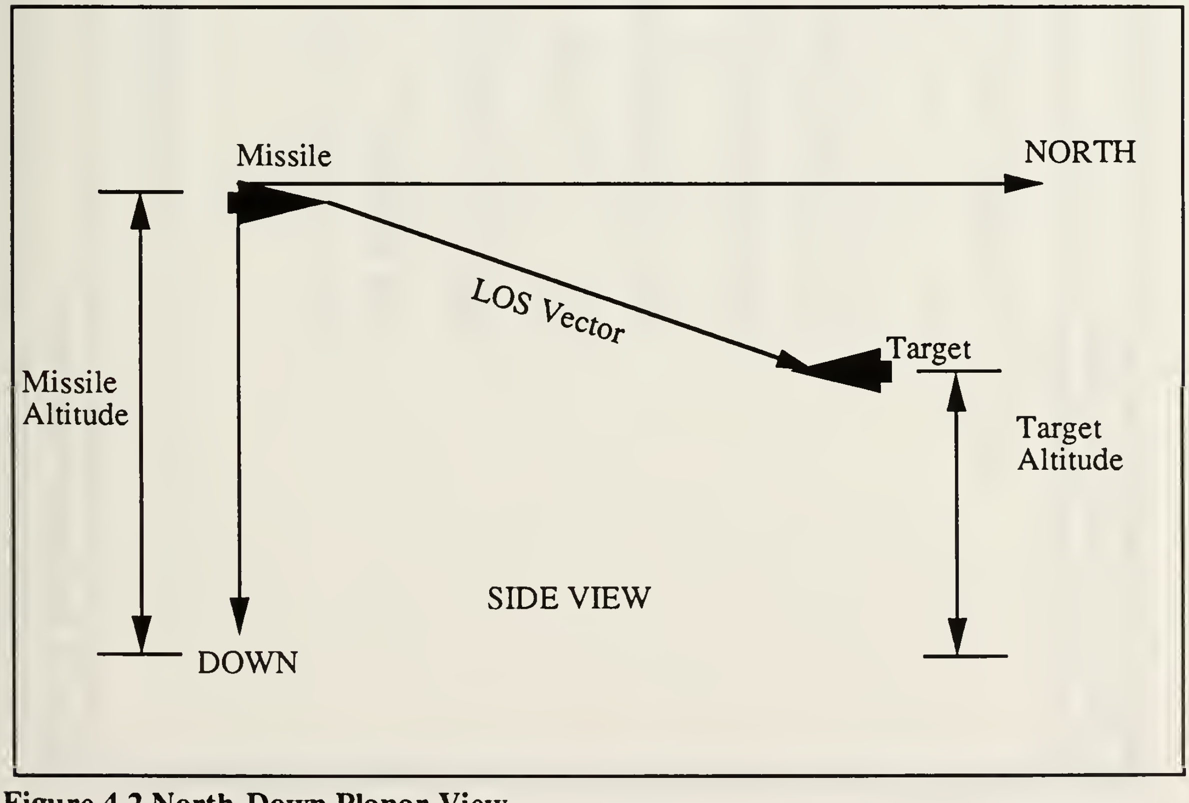
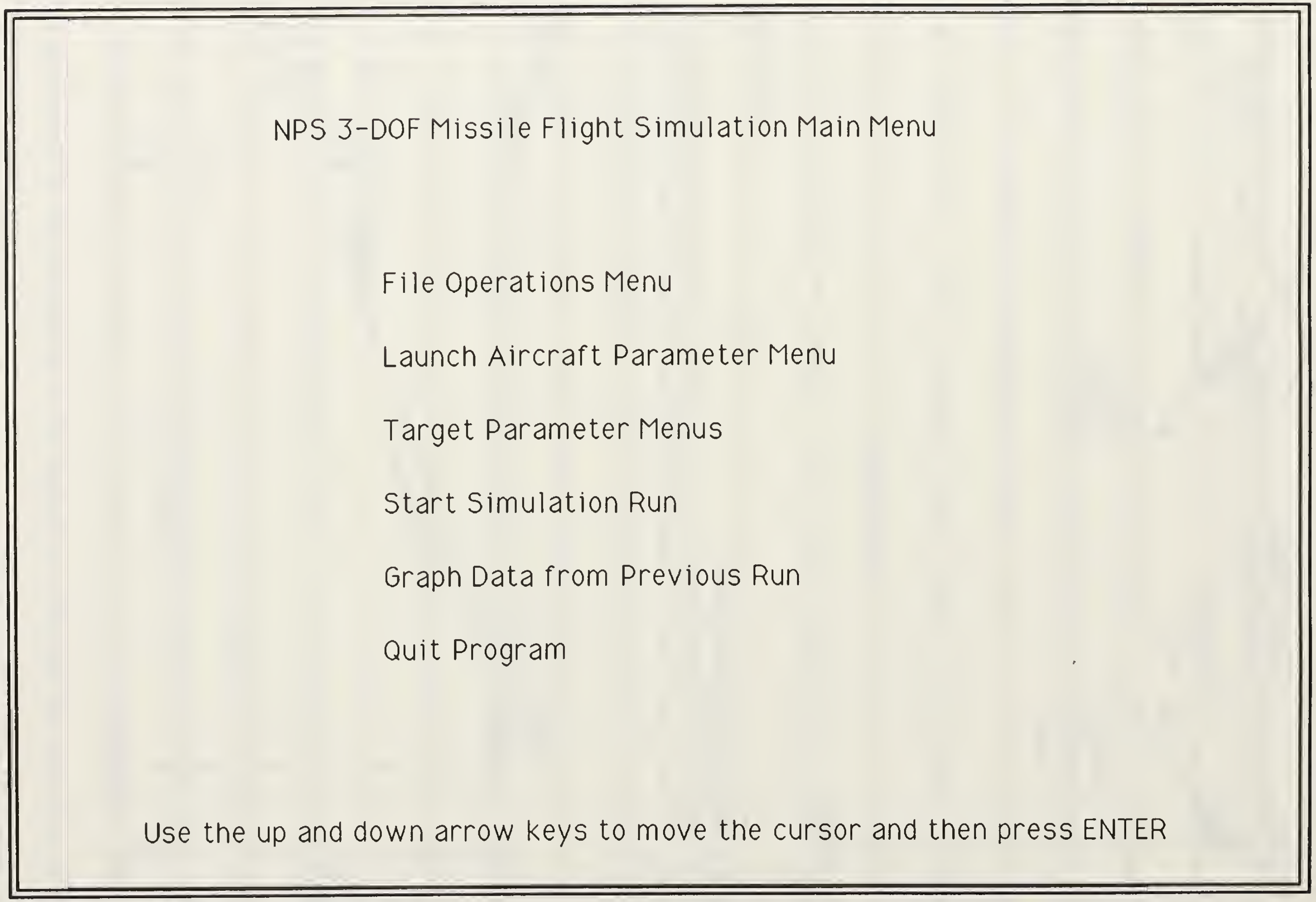
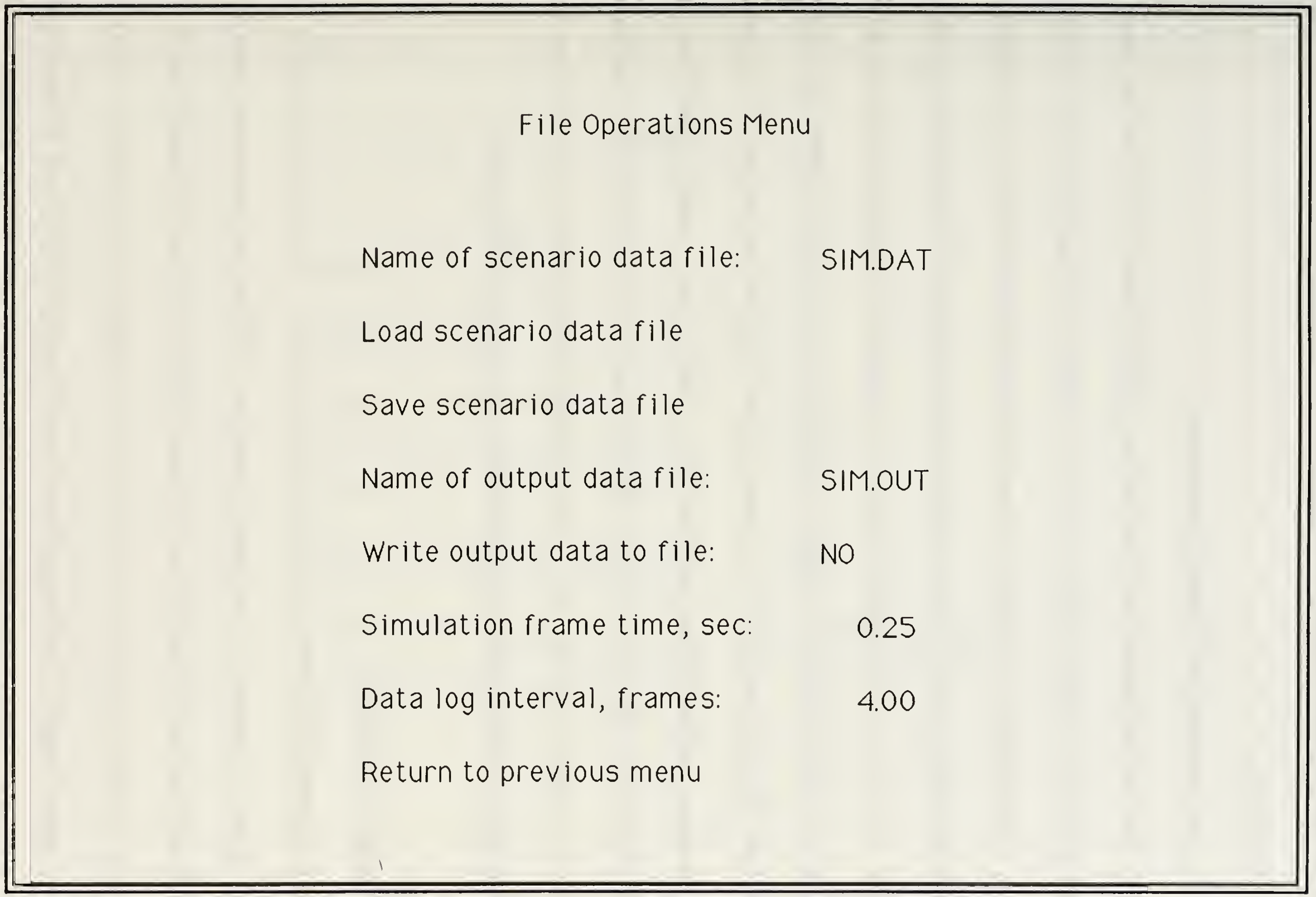


Figure 4.2 North-Down Planar View



```
NPS 3-DOF Missile Flight Simulation Main Menu

     File Operations Menu

     Launch Aircraft Parameter Menu

     Target Parameter Menus

     Start Simulation Run

     Graph Data from Previous Run

     Quit Program

Use the up and down arrow keys to move the cursor and then press ENTER
```

Figure 4.3 Simulation Main Menu



```
File Operations Menu

Name of scenario data file:      SIM.DAT

Load scenario data file

Save scenario data file

Name of output data file:        SIM.OUT

Write output data to file:       NO

Simulation frame time, sec:        0.25

Data log interval, frames:         4.00

Return to previous menu
```

Figure 4.4 File Operations Menu

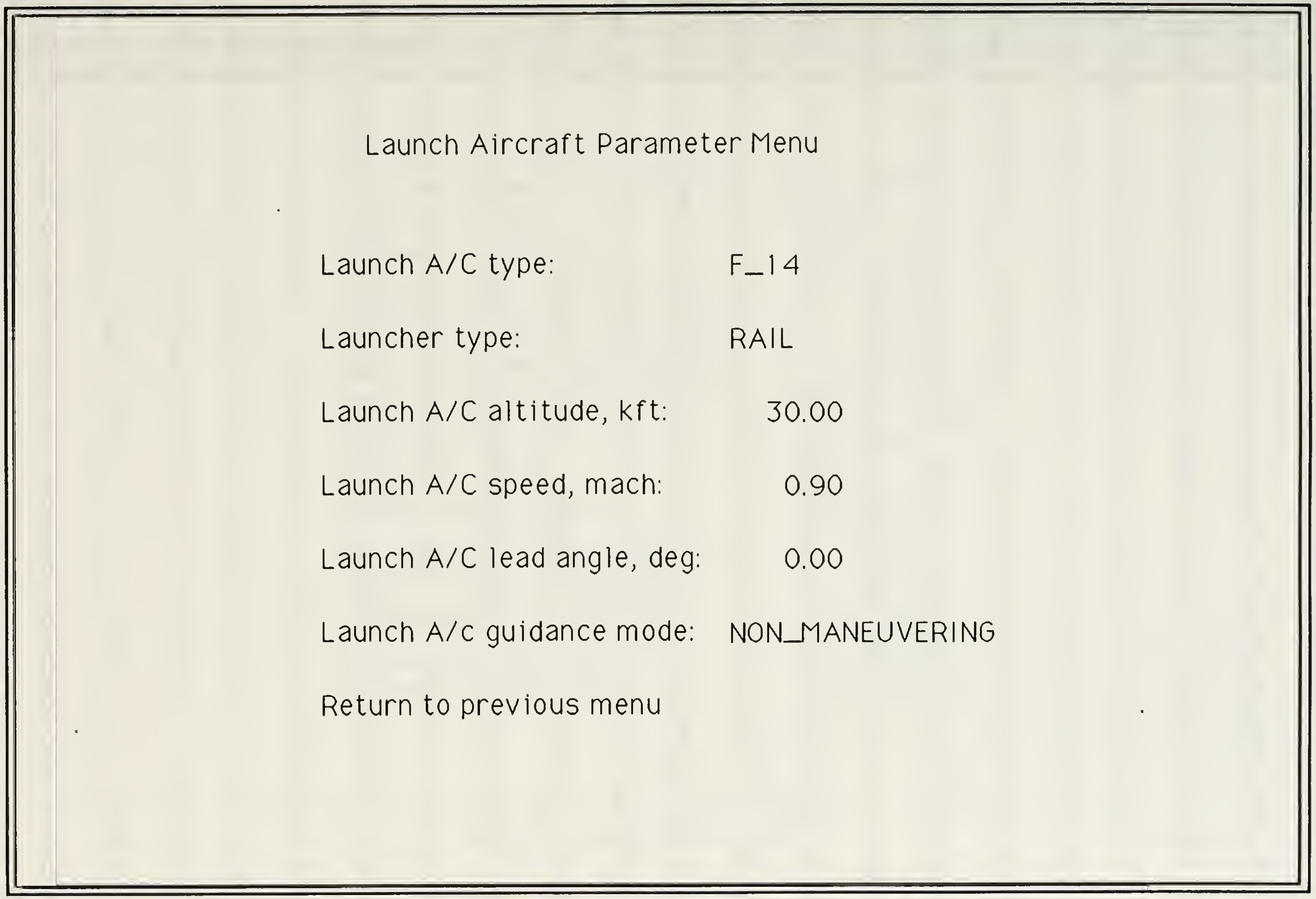
interval. The user may then modify the launch profile by interactively changing the launch aircraft, missile, target, or SOJ parameters. The user does this from their individual menus.

1. Launch Aircraft and Missile Parameters

Launch aircraft parameters that may be modified interactively from the Launch Aircraft Parameter Menu (Figure 4.5) are aircraft type, launcher type, altitude, velocity, heading angle and guidance mode. The launch aircraft's altitude, velocity and heading angle become initial conditions for the missile. The aircraft type, which can be F-14, F-15, or F-18, establishes radar characteristics that will determine when the missile's radar acquires the target. Launcher type specifies whether the missile is launched off a rail station of the launch aircraft or whether the missile is ejected off an ordnance station of the launch aircraft. The guidance modes available to the launch aircraft are the pursuit mode, in which case the launch aircraft maneuvers towards the target after launch or the nonmaneuvering mode, in which case the launch aircraft continues on its original flight path after missile launch.

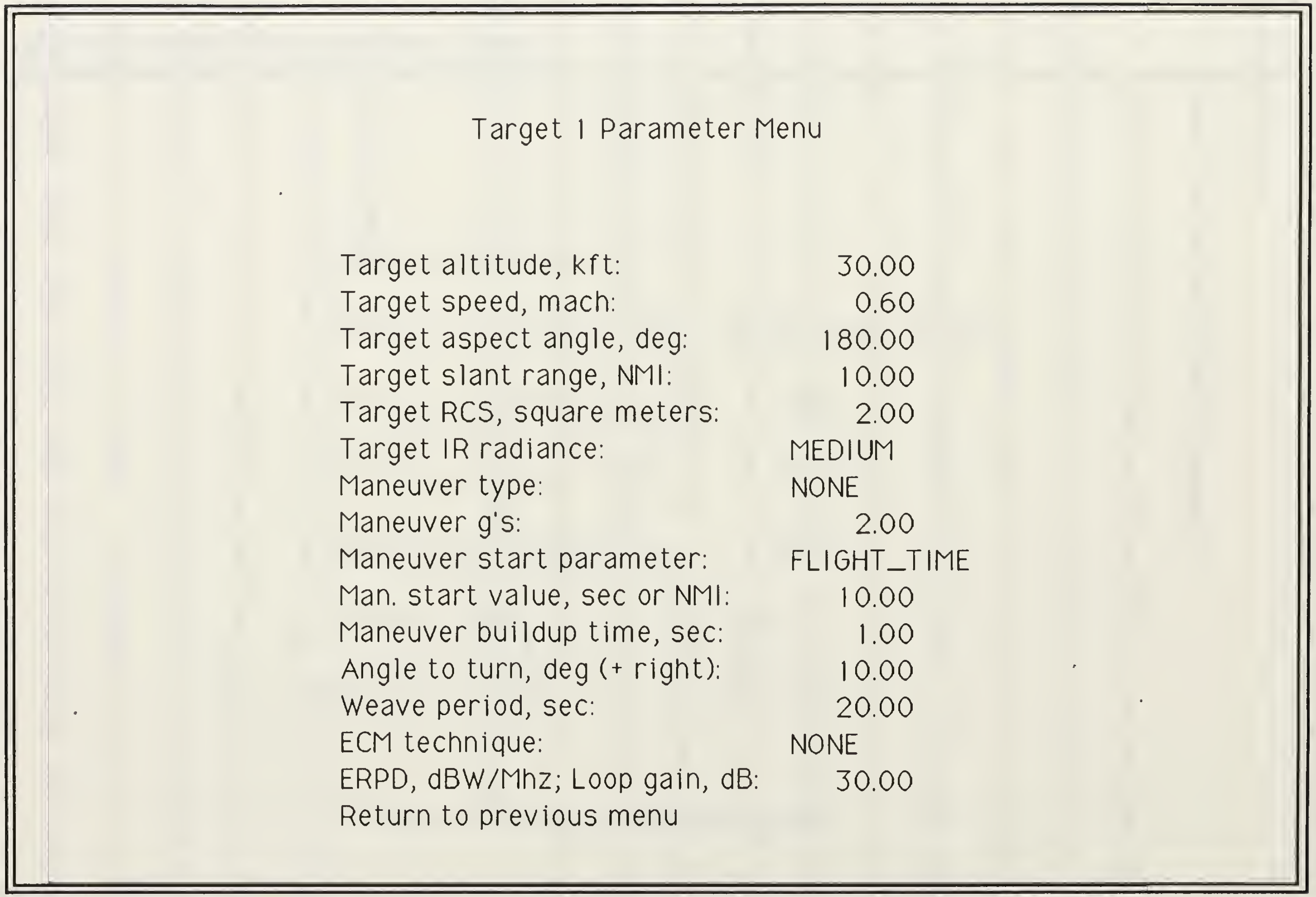
2. Target Parameters

It is assumed that there is always at least one target. The user must enable the second target to simulate a two target formation. The target parameters that may be modified from the Target Parameter Menu (Figure 4.6) can be divided into three categories: general, maneuver, and ECM. The general parameters are altitude, velocity, aspect angle, slant range, radar cross section (RCS), and infrared (IR)



```
            Launch Aircraft Parameter Menu

Launch A/C type:                 F_14

Launcher type:                   RAIL

Launch A/C altitude, kft:          30.00

Launch A/C speed, mach:             0.90

Launch A/C lead angle, deg:         0.00

Launch A/c guidance mode:        NON_MANEUVERING

Return to previous menu
```

Figure 4.5 Launch Aircraft Parameter Menu



```
                 Target 1 Parameter Menu


Target altitude, kft:                 30.00
Target speed, mach:                    0.60
Target aspect angle, deg:            180.00
Target slant range, NMI:              10.00
Target RCS, square meters:             2.00
Target IR radiance:              MEDIUM
Maneuver type:                   NONE
Maneuver g's:                          2.00
Maneuver start parameter:        FLIGHT_TIME
Man. start value, sec or NMI:         10.00
Maneuver buildup time, sec:            1.00
Angle to turn, deg (+ right):         10.00
Weave period, sec:                    20.00
ECM technique:                   NONE
ERPD, dBW/Mhz; Loop gain, dB:         30.00
Return to previous menu
```

Figure 4.6 Target Parameter Menu

radiance. Aspect angle is the angle defined counter-clockwise from the target's tail to the missile's line-of-sight (LOS). Hence, an aspect angle of zero degrees is a tail shot (i.e., the target and launch aircraft have the same heading, with the launch aircraft following the target), and an aspect angle of 180 degrees is a head shot (i.e., the target and missile flying directly at one another). The LOS vector is the vector from the missile's seeker to the target. The slant range is the three dimensional range from the target to the missile. The missile may use its own radar or infrared sensors to provide target information. RCS is a parameter that indicates the targets size in terms of how much radar energy is reflected off the target. A target with a large RCS will be acquired by the missile's radar in a shorter period of time (or similarly acquired at a longer range) than a target with a small RCS. IR radiance is the infrared spectrum's counterpart to RCS, except the IR energy is emitted from the target's engine exhaust (and other IR "hot" spots), rather than being energy reflected off it. The target maneuver parameters control the target's flight path.

The target can be either a nonmaneuvering target, a turning target, or a weaving target depending on the parameter maneuver type. Parameters associated with both the turning target and weaving target are maneuver g's, buildup time, and the maneuver start parameter. Target g's are the number of g's the target is going to achieve executing the maneuver, and build up time is the time it takes to achieve the commanded g's. The maneuver start parameter selects what condition initiates the maneuver. The user can select whether the maneuver starts on flight time (time since the start of the simulation), time remaining (estimated time to intercept), or range-to-go (missile-to-target range). Specific to turn maneuver is the parameter angle to turn through, which is simply how many degrees the target is to turn through before the turn is terminated and the target resumes straight and level flight. A weaving targets flight path resembles a sinusoidal wave form in the north-east plane. That is, looking down on the target, its flight path would resemble a sine wave. The weave specific parameter weave period determines the time (in seconds) it takes the target to complete one cycle or period of the weave. If target two is enabled, the user may define the second targets altitude, the second target's range to target one and the second target's echelon angle (see Figure 4.7).

3. SOJ and Target ECM Parameters

The user also has the option to enable the SOJs and set their range from the launch aircraft, look angle to the target (the angle between the launch aircraft's LOS to the target and the LOS to the SOJ, see Figure 4.8), and modify their ECM parameters (see Figure 4.9). The ECM parameter associated with both targets is ECM technique. The ECM techniques available are none, repeater, and barrage noise. Associated with the repeater ECM technique is the parameter loop gain and associated with the barrage noise ECM technique is the parameter effective radiated power density (ERPD). The repeater technique is intended to deceive the missile by receiving the missile's radar signal, altering it, and retransmitting it back to the missile. The intent is to make the missile "see" the target at a different range or velocity than the target's actual range or velocity. The user may modify the loop gain, effectively controlling the power of the repeater. The barrage noise technique

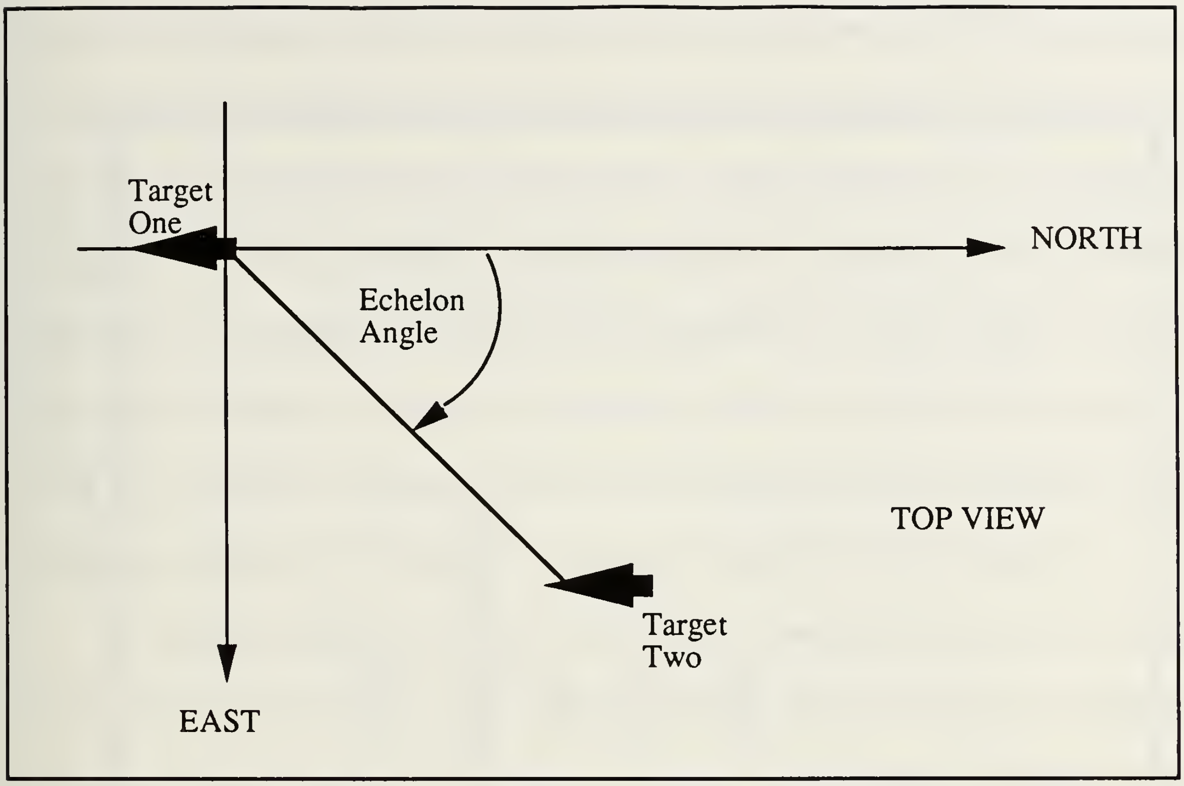


Figure 4.7 Multiple Targets

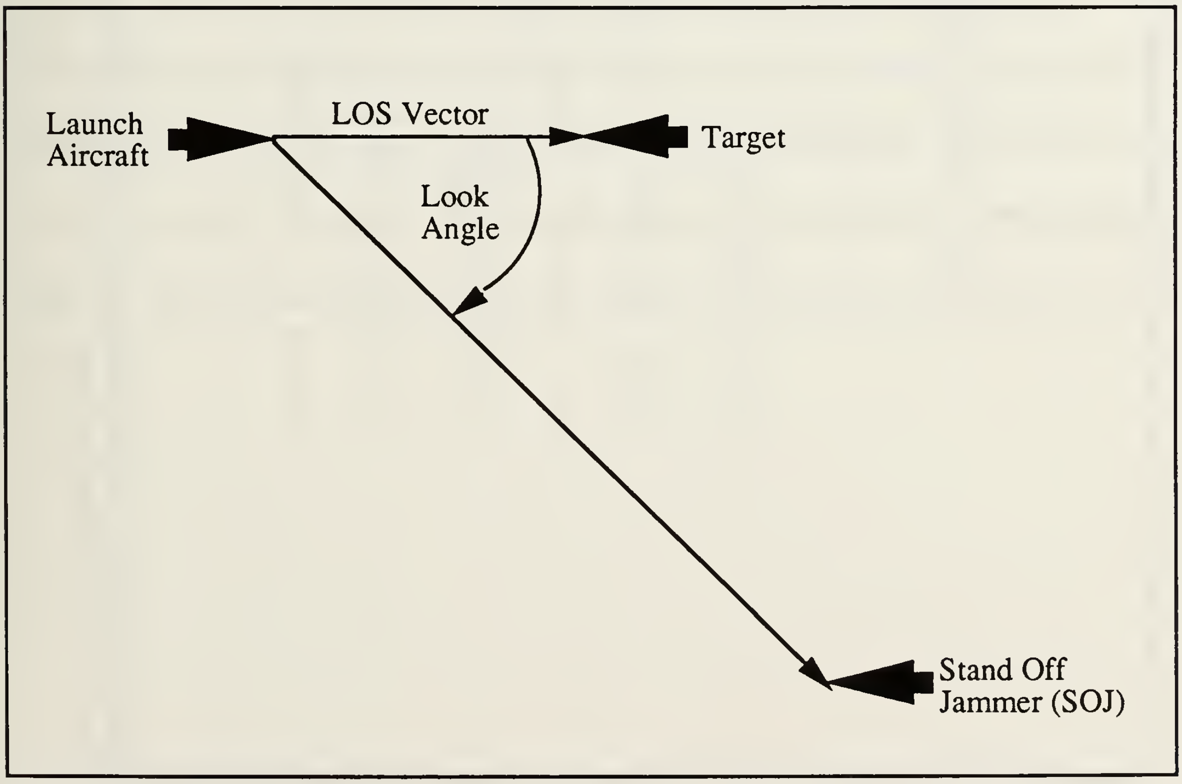
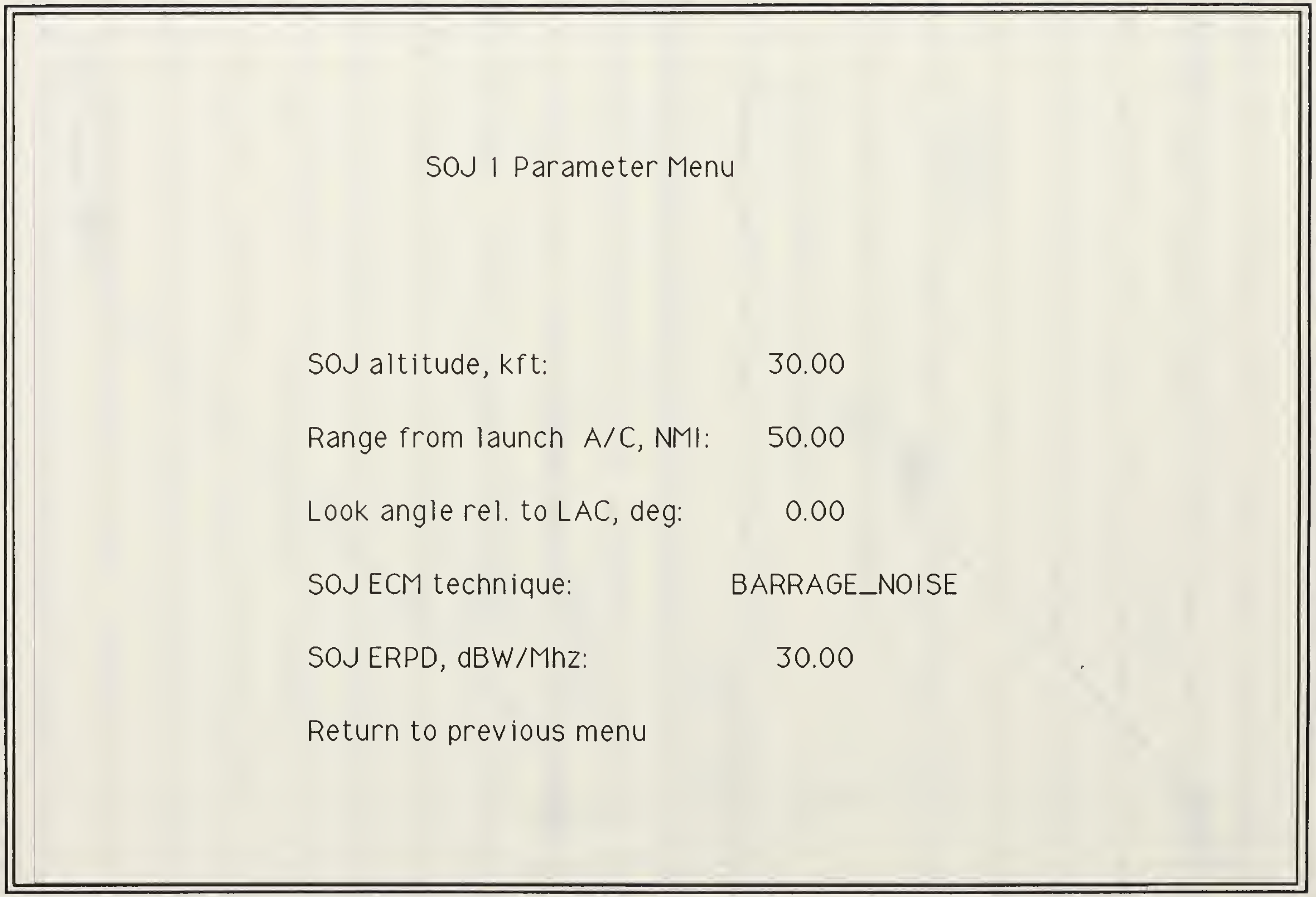


Figure 4.8 SOJ Look Angle



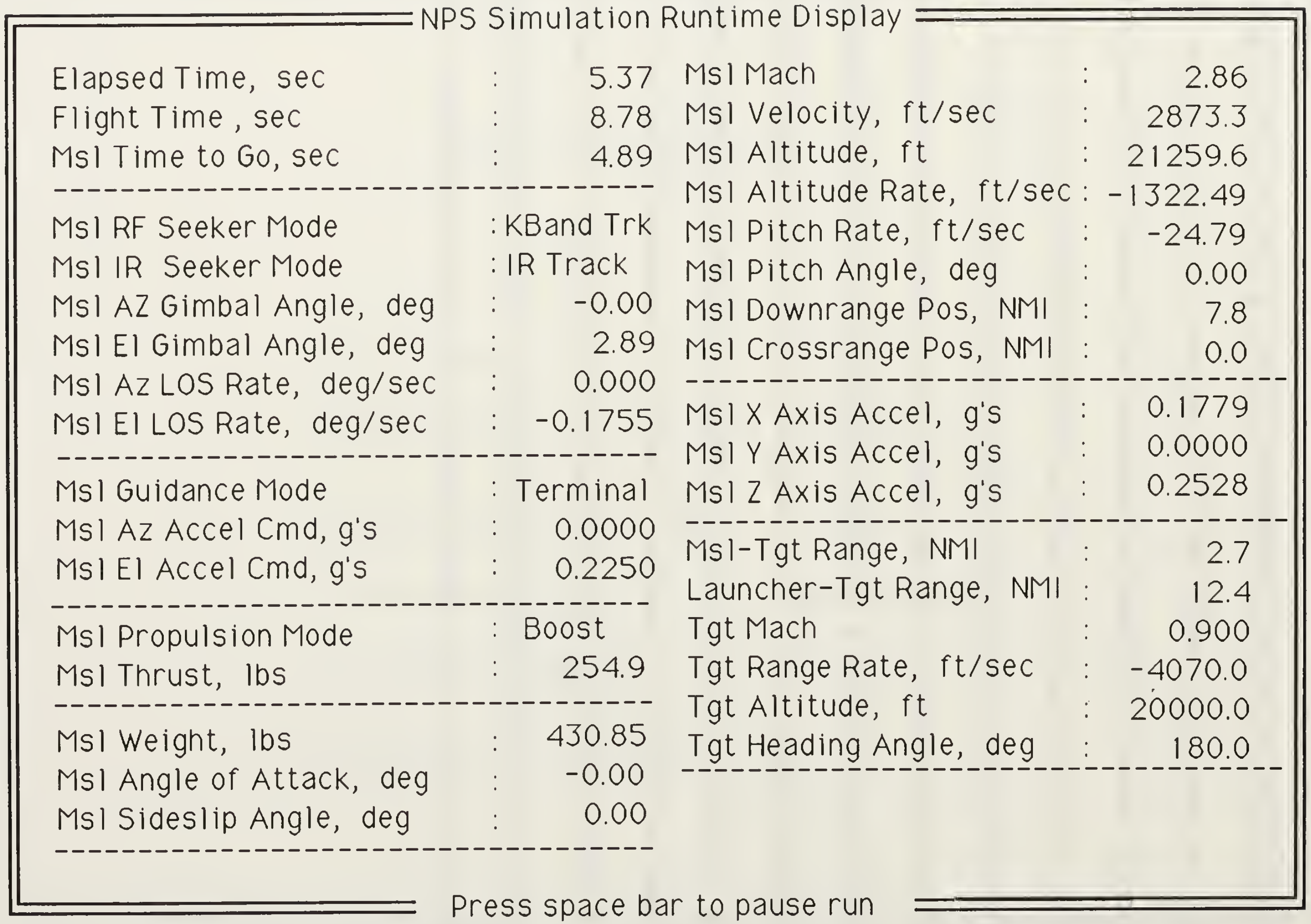
SOJ 1 Parameter Menu

```
SOJ altitude, kft:                  30.00
Range from launch  A/C, NMI:        50.00
Look angle rel. to LAC, deg:         0.00
SOJ ECM technique:               BARRAGE_NOISE
SOJ ERPD, dBW/Mhz:                  30.00
Return to previous menu
```

Figure 4.9 SOJ Parameter Menu

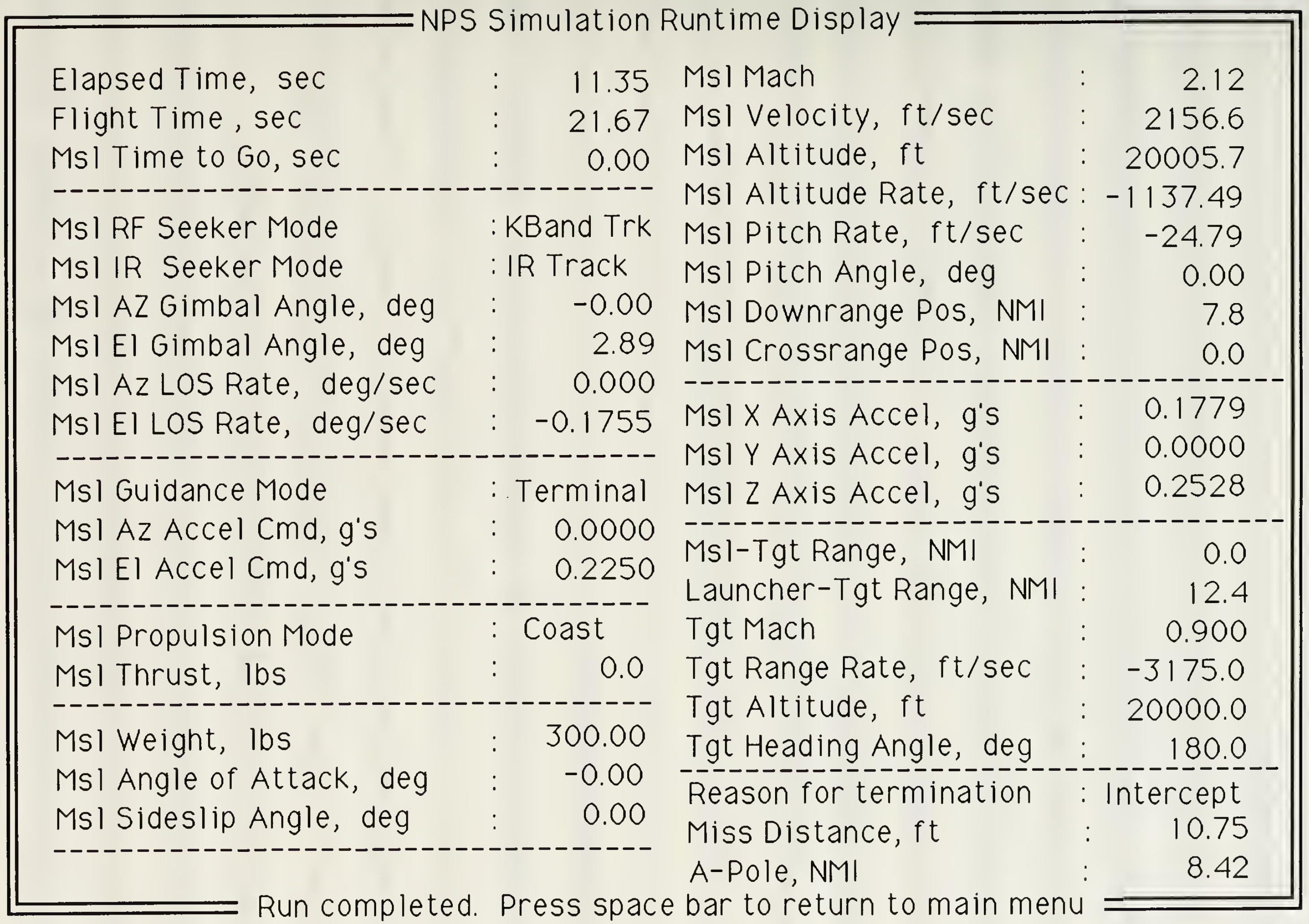
is a cruder technique in which the target transmits broad band noise in an attempt to over-power the missile's radar receiver and deny it any target information. The user may modify the ERPD of the barrage noise. Similar ECM options are available for the SOJs if they are enabled. Basically the ECM is a function of the technique used and the geometry in which the ECM platform (i.e., either target or SOJ) encounters the missile. In all cases just minimal or skeleton code is implemented for ECM due to security classification issues and developmental time constraints. The ECM area falls under the future work category discussed in the last chapter.

To start the simulation, the user selects Start Simulation Run from the main menu. The run-time display is then shown indicating key geometric and kinematic parameters (see Figure 4.10). The user may halt or abort the simulation while it is running. Halting the simulation permits careful examination of parameters during a run, while terminating the simulation allows a quick turn around if the user is not satisfied with the run (e.g., initial conditions set incorrectly). At the end of a simulation run the terminal condition display is shown, which is the runtime display with the final parameter values and the reason for termination (see Figure 4.11).



```
═══════════════════ NPS Simulation Runtime Display ═══════════════════
Elapsed Time,  sec              :      5.37  Msl Mach                      :      2.86
Flight Time , sec               :      8.78  Msl Velocity,  ft/sec         :    2873.3
Msl Time to Go, sec             :      4.89  Msl Altitude,  ft             :   21259.6
-------------------------------------------  Msl Altitude Rate,  ft/sec    :  -1322.49
Msl RF Seeker Mode              : KBand Trk  Msl Pitch Rate,  ft/sec       :    -24.79
Msl IR  Seeker Mode             : IR Track   Msl Pitch Angle,  deg         :      0.00
Msl AZ Gimbal Angle,  deg       :     -0.00  Msl Downrange Pos,  NMI       :       7.8
Msl El Gimbal Angle,  deg       :      2.89  Msl Crossrange Pos,  NMI      :       0.0
Msl Az LOS Rate,  deg/sec       :     0.000  -------------------------------------------
Msl El LOS Rate,  deg/sec       :   -0.1755  Msl X Axis Accel,  g's        :    0.1779
-------------------------------------------  Msl Y Axis Accel,  g's        :    0.0000
Msl Guidance Mode               :  Terminal  Msl Z Axis Accel,  g's        :    0.2528
Msl Az Accel Cmd, g's           :    0.0000  -------------------------------------------
Msl El Accel Cmd, g's           :    0.2250  Msl-Tgt Range,  NMI           :       2.7
-------------------------------------------  Launcher-Tgt Range,  NMI      :      12.4
Msl Propulsion Mode             :   Boost    Tgt Mach                      :     0.900
Msl Thrust,  lbs                :    254.9   Tgt Range Rate,  ft/sec       :   -4070.0
-------------------------------------------  Tgt Altitude,  ft             :   20000.0
Msl Weight,  lbs                :   430.85   Tgt Heading Angle,  deg       :     180.0
Msl Angle of Attack,  deg       :    -0.00   -------------------------------------------
Msl Sideslip Angle,  deg        :     0.00
-------------------------------------------
═══════════════════ Press space bar to pause run ═══════════════════
```

Figure 4.10 Simulation Runtime Display



```
══════════════════ NPS Simulation Runtime Display ══════════════════
Elapsed Time,  sec            :     11.35  Msl Mach                     :      2.12
Flight Time , sec             :     21.67  Msl Velocity,  ft/sec        :    2156.6
Msl Time to Go, sec           :      0.00  Msl Altitude,  ft            :   20005.7
-------------------------------------      Msl Altitude Rate,  ft/sec   : -1137.49
Msl RF Seeker Mode            : KBand Trk  Msl Pitch Rate,  ft/sec      :   -24.79
Msl IR  Seeker Mode           : IR Track   Msl Pitch Angle,  deg        :      0.00
Msl AZ Gimbal Angle,  deg     :     -0.00  Msl Downrange Pos,  NMI      :       7.8
Msl El Gimbal Angle,  deg     :      2.89  Msl Crossrange Pos,  NMI     :       0.0
Msl Az LOS Rate,  deg/sec     :     0.000  ------------------------------------------
Msl El LOS Rate,  deg/sec     :   -0.1755  Msl X Axis Accel,  g's       :    0.1779
-------------------------------------      Msl Y Axis Accel,  g's       :    0.0000
Msl Guidance Mode             :  Terminal  Msl Z Axis Accel,  g's       :    0.2528
Msl Az Accel Cmd, g's         :    0.0000  ------------------------------------------
Msl El Accel Cmd, g's         :    0.2250  Msl-Tgt Range,  NMI          :       0.0
-------------------------------------      Launcher-Tgt Range,  NMI     :      12.4
Msl Propulsion Mode           :  Coast     Tgt Mach                     :     0.900
Msl Thrust,  lbs              :       0.0  Tgt Range Rate,  ft/sec      :   -3175.0
-------------------------------------      Tgt Altitude,  ft            :   20000.0
Msl Weight,  lbs              :    300.00  Tgt Heading Angle,  deg      :     180.0
Msl Angle of Attack,  deg     :     -0.00  ------------------------------------------
Msl Sideslip Angle,  deg      :      0.00  Reason for termination       : Intercept
-------------------------------------      Miss Distance, ft            :     10.75
                                           A-Pole, NMI                  :      8.42
═══════════ Run completed.  Press space bar to return to main menu ═══════════
```

Figure 4.11 Simulation Termination Display

V. THE CONTROL AND SUPPORT OBJECTS

A. INTRODUCTION

This chapter presents three of the top-level objects that control the simulation and their support objects (see Figure A.1 of Appendix A). The discussion of the top-level objects will follow the basic development methodology presented earlier and illustrates how the objects demonstrate important features of object oriented principles and software engineering principles. This discussion is not intended to be a line by line detailed functional description of the code, but is intended to provide a basic understanding of how the simulation works (the reader is referred to Appendix C and Appendix D for a complete listing of the simulation).

B. BASIC METHODOLOGY

Objects and their messages will appear in capitol letters by convention through the remainder of this document. The methodology presented in Chapter II will be used to develop the three top-level objects presented in this chapter. These top-level objects, and their support objects, will be discussed here to provide a framework for presenting the MISSILE and TARGET objects in the next chapter.

At this the highest level of abstraction, we wish to keep the simulation application independent. We might wish to simulate a power plant or missile - our upper most level should not reflect what particular application we are using. Objects are necessary to control or manage the simulation. By controlling or managing the simulation we mean things like getting user input, initialization, starting and stopping the simulation, and presenting data. The objects necessary for these operations are identified as the EXECUTIVE, APPLICATION, and USER_INTERFACE. The EXECUTIVE has no knowledge of what type of simulation is running, it just sees the APPLICATION and USER_INTERFACE objects and stimulates them with the appropriate messages. The APPLICATION object has the knowledge of the specific details of the application in terms of what objects exist and their interfaces. The USER_INTERFACE object is required for user input and output. The APPLICATION will have to be visible to the USER_INTERFACE and the USER_INTERFACE must be visible to the APPLICATION. The reasons for this will become clear when we examine each object's interface or messages. So far we have identified the objects, formed a general characterization of their behavior and established their visibility. The next steps of the development methodology, defining the interfaces and implementing the objects, will be discussed in more detail in the following sections.

C. OBJECT MESSAGES AND IMPLEMENTATION

Each object's messages are presented in more detail because they are key to understanding an object's capabilities. The Ada with clause allows a package (or object) to access or view another package's specification. Package specifications define the interface to the package in terms of data structures, function calls and procedure calls available to the users of the package. In our object oriented view, package specifications define the external messages that an object can respond to by eliciting some type of action or providing the sender with information. Internal messages are the functions and procedures that are in the body and not in the package specification, and therefor are for the exclusive use of that object. Appendix A provides figures that illustrate the objects indicating which objects they "with" and their messages.

D. THE CONTROL OBJECTS

1. The EXECUTIVE

The simulation EXECUTIVE is a procedure that forms the upper or outer-most layer of the simulation (Figure A.2). This is highest level of abstraction for the simulation. The simulation EXECUTIVE contains a context clause that "withes" the APPLICATION object and the USER_INTERFACE object. The EXECUTIVE sends a message to APPLICATION to initialize the system and a message to the USER_INTERFACE to turn over control of the simulation to the user.

2. The APPLICATION

The APPLICATION also resides at the simulation's highest level of abstraction. The APPLICATION object contains or defines the application specific messages or actions for a specific simulation. The APPLICATION might represent one of any number of simulations. The APPLICATION object has the following messages as shown in Figure A.3: INITIALIZE_SYSTEM, INITIALIZE_SIMULATION, NUMBER_OF_STATE_VARIABLES, GET_STATES, PUT_STATES, COMPUTE_DERIVATIVES, LOG_DATA, END_CONDITION_MET, END_OF_RUN, CHECK_PAUSE, and SIMULATION_MAIN. The internal INITIALIZE_SYSTEM message sends a message to the SYSTEM_SPECIFIC object to initialize the video display and sends a message to the USER_INTERFACE to display the initial title screen to the user. INITIALIZE_SIMULATION ,an internal message, signals the objects that make up the specific application to initialize themselves. This message also frees memory by instantiating Ada's UNCHECKED_DEALLOCATION to create a procedure called FREE. FREE deallocates memory that was used for storing the previous runs data. INITIALIZE_SIMULATION will also create an output data file if the simulation is logging data to disk. The external COMPUTE_DERIVATIVES message is invoked which in turn sends COMPUTE messages to the MISSILE and TARGET objects telling them to compute the mathematical derivatives that characterize them. These are used as initial values for computing the state variables in the first time step. An example of sending or invoking the INITIALIZE_SYSTEM from another object (e.g., as is done in the body of EXECUTIVE for this particular message) is as follows:

```
APPLICATION.INITIALIZE_SYSTEM
```

When this statement is encountered in the body of EXECUTIVE the INITIALIZE_SYSTEM message or procedure defined in APPLICATION will be executed. The messages are intended to be self-descriptive.

GET_DERIVATIVES is an external message that provides the values of the MISSILE and TARGET derivatives by sending a GET_DERIVATIVES messages to these objects. The external NUMBER_OF_STATE_VARIABLES message returns the number of state variables possessed by a specific application. This message is used to correctly size the data structures in the INTEGRATION object. The external message GET_STATES solicits the appropriate objects for their state variables while PUT_STATES provides those objects with updated state variables.

The internal LOG_DATA message tells the simulation to output data to the screen and, if desired, save data to a disk file. The internal CHECK_PAUSE message checks to see if the user has paused the simulation run or has decided to terminate the run. The internal END_CONDITION_MET message signals that the appropriate conditions have been satisfied and the simulation can be terminated.

The external message SIMULATION_MAIN is the heart of the APPLICATION object. The first action SIMULATION_MAIN takes is to set the Boolean variable LOGGING_TO_DISK according to the value found in SETUP_VALUES. If LOGGING_TO_DISK is true, data generated by the simulation will be written to a file on disk. SIMULATION_MAIN then tells the objects to set themselves up by sending a SETUP message to the missile and aircraft objects. There is a distinction in the simulation between setup and initialize. Setup refers to collecting data, either interactively from the user or from default values, while initialize refers to using these values to compute initial conditions or initialize data structures. After the MISSILE and TARGET have been setup, a message is passed to the ENVIRONMENT object to set the time to zero. This is the reference time at which the simulation starts. Following this a message is sent to the INTEGRATION object establishing the TIME_STEP_SIZE to be used for the numeric integration. The NEXT_LOGGING_FRAME and FRAME_NUMBER are then set to initial values. Then the simulation is initialized followed by the INTEGRATION object. At this point the main loop is entered. This body of code will be repeatedly executed until the simulation stops, either by reaching normal end conditions or through user intervention. Within the loop the INTEGRATION object is told to ADVANCE_TIME. This is the message that drives the computation of state variables and hence most of the computations or activity taking place in the simulation. Also within the loop, the screen is updated and data logged to disk if required. The loop is repeatedly executed until END_CONDITION_MET is true. END_CONDITION_MET sends a message to the MISSILE to see if the MISSILE specific END_CONDITION_MET is true (e.g., TARGET intercept has occurred, the MISSILE has flown into the ground, etc.) or checks to see if the user has terminated the run. Once END_CONDITION_MET is true, the run-time screen is updated for the last time and the internal SHOW_TERMINAL_CONDITIONS message displays the terminal conditions of the flight to the user. Finally, the internal END_OF_RUN message is invoked to close the output data file if the simulation was logging data to disk. The user then can return to the main menu. At this point the simulation may be run again or parameters may be modified.

3. The USER_INTERFACE

The USER_INTERFACE object, shown in Figure A.4, allows the user to the user to control the simulation through keyboard input, along with presenting run-time displays and simulation status information to the user. The USER_INTERFACE is designed as a standard interface regardless of what model computer hosts the simulation. USER_INTERFACE is menu driven to provide a more "user/programmer friendly" interface than previous FORTRAN simulations. USER_INTERFACE uses some of Ada's modern language features that are not found in FORTRAN. For example, access types (pointers) and records are used to form linked lists. The linked lists that form the menus, submenus, and individual items are quite easily modified to accommodate growth (i.e., more menus ,submenus, or items). Recursion is used to traverse the lists. Recursion makes the code easier to read and understand. Ada's variant records simplified the design and building of the linked lists.

The MATH, APPLICATION and SYSTEM_SPECIFIC objects are made visible to USER_INTERFACE by with clauses. USER_INTERFACE also "withes" the Ada predefined packages TEXT_IO and REAL_IO. USER_INTERFACE provides the following external messages to its users: MAIN, SHOW_TITLE_SCREEN, DRAW_RUNTIME_BORDER, and

SETUP_RUNTIME_SCREEN. The MAIN message passes an access type to the MAIN_MENU as a parameter to the internal MANAGE_MENU object, the heart of USER_INTERFACE. MANAGE_MENU navigates through the various menus, submenus, and individual items. MANAGE_MENU allows the user to interactively enter or modify data. The menus allow the user to build a "missile launch scenario" by entering or selecting missile and target parameters. The user may enter a value for the missile's launch altitude by typing it at the keyboard, for example. Where data takes the form of an enumerated type, for example the launcher type is either F-14, F-15, or F-18, the user can cycle through the choices by striking the enter key. This is implemented by using the PRED (for predecessor) or SUCC (for successor) attributes of enumerated types.

The user can initiate three actions from USER_INTERFACE: LOAD_DATA_FILE, SAVE_DATA_FILE, or START_RUN. LOAD_DATA_FILE provides the missile launch scenario parameters from a disk file. SAVE_DATA_FILE will save the current missile launch scenario, possibly customized by the user, to a disk file for later use. The intent here is to enable the quick setup of missile launch scenarios that vary from the default scenario. START_RUN will gather all the missile/target parameters from the various menus. These values become the appropriate object's setup values. At this point START_RUN sends a message to APPLICATION (i.e., APPLICATION.MAIN(SETUP_VALUES)) that starts the simulation.

The SHOW_TITLE_SCREEN external message clears the user's screen, turns off the cursor and displays the initial welcome text. DRAW_RUNTIME_BORDER draws the screen border for the runtime screen and displays a text instruction. The message SETUP_RUNTIME_SCREEN clears the screen and signals DRAW_RUNTIME_BORDER. Then DRAW_RUNTIME_SCREEN displays a template of the text portion of the simulation data, with the appropriate units, that is presented during runtime. For example, "Elapsed time, sec:" is displayed. The actual run-time data values are displayed from the APPLICATION object.

Other important internal messages are: DISPLAY_MENU, SETUP_MENU_DATA, GET_TEXT, and GET_REAL. DISPLAY_MENU is a message that displays the individual items in a menu or submenu. This message uses the SYSTEM_SPECIFIC object, which will be discussed shortly, for low-level IO. SETUP_MENU_DATA establishes all the default values for the MISSILE and TARGET parameters found in the menus. GET_TEXT is used to display a prompt for text input and then input and validate the user's response. GET_TEXT also uses the SYSTEM_SPECIFIC object for much of its low-level IO. GET_REAL is used to read real values input by the user. This message provides a more flexible and user-friendly method for inputting real numbers than Ada's predefined REAL_IO package. SETUP_MENU_DATA, SHOW_TITLE_SCREEN, and SETUP_RUNTIME_SCREEN are implemented as separate compilation subunits of the main USER_INTERFACE package. This division was made because these units contain mostly textual information that tended to clutter and obscure the main

USER_INTERFACE package. Also, modifications are easier and compilations are faster when these messages are implemented as subunits of the USER_INTERFACE package. Editing is easier with smaller modules and the corresponding compilation faster.

E. THE SUPPORT OBJECTS

1. SYSTEM_SPECIFIC

Modular design and information hiding allow the simulation to be machine independent. The simulation was developed and implemented on an IBM AT compatible machine. In the future the simulation will be modified to run on an Apple MacIntosh computer and possibly other systems. To aid this process, all the machine dependent code is implemented (hidden) in the SYSTEM_SPECIFIC object (see Figure A.5). Most of this code is associated with the IBM video display. By rewriting SYSTEM_SPECIFIC for the Apple MacIntosh, and keeping the original message names, the porting process should consist simply of a recompile of SYSTEM_SPECIFIC and a link of the simulation. Also by working at a lower system-specific level all screen displays are output in the most efficient manner providing very fast screen updates. This prevents the user from perceiving a delay as the screen is updated or the next menu is displayed (problems experienced in earlier FORTRAN simulations).

SYSTEM_SPECIFIC is visible to APPLICATION and USER_INTERFACE. A number of packages providing DOS environmental support are included with Meridian's Ada compiler [Ref. 13]. These include: SYSTEM, PROGRAM_CONTROL, INTERRUPT, COMMON_DISPLAY_TYPES, TTY and BOX. The SYSTEM package is used to provide an address expression which is a 32 bit segmented memory address. The address expressions are used for monochrome and color video addresses. The PROGRAM_CONTROL package is used to terminate the simulation. PROGRAM_CONTROL's QUIT procedure terminates the calling program and returns control to DOS. The package INTERRUPT allows calls to DOS interrupt vectors. COMMON_DISPLAY_TYPES contains declarations for the various packages that handle display operations, such as TTY and BOX. The package TTY provides operations on the terminal display and keyboard. TTY links in faster than TEXT_IO and calls to the TTY subprograms run faster. The TTY subprograms used are GET, PUT, and CHAR_READY. CHAR_READY determines if a character is ready to be read from the keyboard. Package BOX provides procedures for drawing boxes on the text screen. Also, a Meridian supplied package, BIT_OPS, is used for bit-level logical operations [Ref. 14].

All of these vendor supplied packages are used in the SYSTEM_SPECIFIC to provide the following external messages: INIT_VIDEO, DRAW_BOX, CLEAR_SCREEN, REVERSE_VIDEO_ON, REVERSE_VIDEO_OFF, MOVE_CURSOR, TURN_CURSOR_ON, TURN_CURSOR_OFF, PUT_STRING, PUT_REAL, INPUT_STRING, KEY_AVAILABLE, GET_KEY, GET_MENU_COMMAND. INIT_VIDEO determines whether the host system has a monochrome or EGA display and sets variables accordingly. DRAW_BOX is used to draw the screen border. CLEAR_SCREEN clears the video display. REVERSE_VIDEO_ON and REVERSE_VIDEO_OFF control whether text is output in reverse video. MOVE_CURSOR moves the cursor to the desired row and column of the video display. TURN_CURSOR_ON and TURN_CURSOR_OFF control whether the cursor is displayed. PUT_STRING outputs a character string to the video display in normal or reverse video. PUT_REAL outputs a real number to the screen. INPUT_STRING inputs text strings from the user. KEY_AVAILABLE indicates if a keyboard key has been pressed and GET_KEY returns the scan code of the key pressed. GET_MENU_COMMAND uses KEY_AVAILABLE and GET_KEY to decode the keyboard input into UP_ARROW, DOWN_ARROW and ENTER commands. The implementation details involve advanced DOS programming and the reader is referred to Young [Ref. 15] for further information.

2. INTEGRATION

The INTEGRATION (see Figure A.6) object performs the numerical integration of the MISSILE and TARGET state variables. INTEGRATION is visible to the APPLICATION object. Objects visible to INTEGRATION are MATH, ENVIRONMENT, and APPLICATION. The integration of the state variables each time step drives the MISSILE and TARGET in the simulation. Each TIME_STEP time units, the state variables are integrated and fed back to the appropriate objects and, along with other computations, are used to form the subsequent TIME_STEP's state variables. For example, INTEGRATE requests the MISSILE's derivatives, which are the MISSILE's accelerations and velocities. These accelerations and velocities are computed by the MISSILE using the previous TIME_STEP's state variables (along with other computations). INTEGRATION then integrates the MISSILE's accelerations and velocities to obtain the MISSILE's velocities and position, which form the MISSILE's current state variables. These state variables are then provided to the MISSILE for use during the next TIME_STEP, time is advanced TIME_STEP units, and the whole process is repeated.

The external messages that make all this possible are TIME_STEP_SIZE, SET_TIME_STEP_SIZE, INITIALIZE, and ADVANCE_TIME. These messages form a standard interface regardless of the specific application. TIME_STEP_SIZE provides the sender with the current value INTEGRATION is using for TIME_STEP. SET_TIME_STEP_SIZE permits INTEGRATION's TIME_STEP value to be changed. INITIALIZE requests that APPLICATION return the application specific object's derivatives and state variables. These values are used as initial conditions by INTEGRATION. ADVANCE_TIME is the heart of INTEGRATION, as it signals the correct integration method to execute. There are numerous methods to perform numeric integration, for example, Hanna, Euler, Adams-Bashforth, to name just a few. These methods offer trade offs in terms of accuracy and execution speed. By having the method visible only to ADVANCE_TIME, changing the particular method used is relatively easy. At this time only the Hanna method is implemented [Ref. 16]. The Hanna method is a predictor/corrector numeric integration technique. A temporary state or predictor, which is the previous TIME_STEP's state variables, multiplied by its derivatives and TIME_STEP, is computed. Next the ENVIRONMENT object, which keeps track of time, is signalled to increment the current time by TIME_STEP units by the SET_TIME message. Then INTEGRATION sends MISSILE and TARGET their temporary state variables via a message to APPLICATION. INTEGRATION then signals MISSILE and TARGET to compute their derivatives via another message to APPLICATION. MISSILE and TARGET use their temporary state variables to compute their current derivatives. INTEGRATION then requests these derivatives to correct the state variables. Finally the current set of derivatives is saved for the next TIME_STEP. This process is repeated each time INTEGRATION is sent the ADVANCE_TIME message by APPLICATION until APPLICATION stops the simulation.

3. MATH

The MATH object, shown in Figure A.7, provides external messages that perform all the basic mathematical operations required by the simulation. MATH also defines all the physical constants used in the simulation, such as PI, E, and the gravitational constant G. MATH is visible to most of the upper-level objects. MATH withes Meridian's MATH_LIB to provide trigonometric functions such as SIN, COS, TAN and their inverses. All these functions have been embedded in functions that perform a type conversion of the operands to the real type, as the Meridian MATH_LIB is instantiated for the float type. MATH instantiates the real types. Operations, in addition to those provided by MATH_LIB, include LOG, LIMIT, **, MIN, MAX and a variety of matrix and vector operations. LOG provides the base 10 logarithm of a number. The overloaded LIMIT compares a real variable against a lower and upper limit, and returns either the upper limit if the variable is greater than it, the lower limit if the variable is less than it, or the variable, if it falls between the two limits. The other LIMIT compares a real variable with the positive and negative values of a single limit and returns the appropriate value. The ** message provides exponentiation. MIN returns the minimum of two variables while MAX returns the maximum of two variables. MATH also instantiates REAL_MATRIX from the generic unit MATRIX_AND_VECTOR to provide mathematical operations on matrices and vectors.

4. REAL_MATRIX

REAL_MATRIX (see Figure A.7) is an instantiation of the generic unit MATRIX_AND_VECTOR and illustrates Ada's limited implementation of the object oriented inheritance concept. MATRIX_AND_VECTOR is a generic unit that acts as a template for packages and provides the means to build reusable software components. MATRIX_AND_VECTOR provides all the basic mathematical operations for matrices and vectors without specifying what data type make up these structures. REAL_MATRIX is the MATRIX_AND_VECTOR object instantiated for the real data type. Future requirements may call for the instantiation of MATRIX_AND_VECTOR for complex numbers.

REAL_MATRIX provides a number of messages for operations on matrices and vectors because many of the quantities encountered in the simulation, such as forces, are best expressed in terms of vectors or arrays. REAL_MATRIX defines the vector type as a one dimensional array of real numbers and the matrix type as a two dimensional array of real numbers. The basic operations on matrices are overloaded to deal with both single and two dimensional matrices. Ada's attributes for array types were very useful in coding these operations. The RANGE attribute provided an easy and flexible method for specifying array index constraints. The LAST attribute proved valuable for specifying the upper bound on the control variable of for loops. The overloaded +, -, * and /, provide for the addition, subtraction, multiplication and division of arrays respectively. These messages also contribute to the readability of the code. MAGNITUDE returns the magnitude of a vector and CROSS_PRODUCT provides the cross product of two vectors. The matrix messages TRANSPOSE, IDENTITY, DETERMINANT, and INVERSE provide the services that their names suggest [Ref. 17].

VI. THE MISSILE, LAUNCHER AND TARGETS OBJECTS

A. INTRODUCTION

This chapter discusses the missile, launcher, and target objects. One of our major software engineering goals, understandability, is achieved by implementing and discussing the core of the simulation in terms of modular objects. This approach also serves to accurately map the real-world problem space (see Figure 3.1) into the objects that form the software solution, illustrated in Figure B.1 of Appendix B.

B. THE MISSILE

The MISSILE object or package withes the LAUNCHER, TARGETS, missile subsystem objects and support objects (see Figure B.2). The LAUNCHER object provides the missile with launch aircraft information and TARGETS provides target information. Missile subsystem objects are the AIRFRAME, AUTOPILOT, RF_SEEKER, IR_SEEKER, and GUIDANCE. The AIRFRAME contains further subsystems such as the AERO and THRUST. The missile subsystems serve as a good example of abstraction, modularity, low coupling and high cohesion. The support objects are MATH, INTEGRATION, KINEMATICS, and ENVIRONMENT. MISSILE uses the time keeping messages of ENVIRONMENT.

1. MISSILE Messages

MISSILE messages include SETUP, INITIALIZE, PUT_STATES, GET_STATES, GET_DERIVATIVES, LOG_DATA, MANEUVER_VALUE, END_CONDITIONS_MET, TERMINAL_CONDITIONS, and COMPUTE. SETUP establishes the launch type, number of targets, number of SOJs and ECM power . SETUP also signals the RF_SEEKER and KINEMATIC objects to proceed with their SETUP routines. The INITIALIZE message establishes initial conditions for many of the missile's physical characteristics such as missile mass, drag, thrust, and initial phase of flight. Also during initialization geometric initial conditions such as ranges and heading angle are computed. AIRFRAME, GUIDANCE, RF_SEEKER, and IR_SEEKER are also signaled to initialize.

The message PUT_STATES accepts the new states (i.e, missile position and velocity) when signaled by the APPLICATION object. GET_STATES presents the caller with the current missile states and GET_DERIVATIVES presents the derivatives of the current states (i.e., missile velocity and acceleration).

The LOG_DATA message provides all the missile data, approximately forty two items, that are logged to disk or presented on screen to the user. MANEUVER_START_VALUE returns the time since launch, time-to-go or range-to-go, which is used to initiate the target maneuver. END_CONDITIONS_MET provides a Boolean value indicating whether the appropriate conditions (e.g, target intercept, missile physical limits exceeded, out of energy, flew into the ground) have been met to terminate missile flight. TERMINAL_CONDITIONS supplies the reason for flight termination plus terminal data items of interest such as miss distance (defined as the point of closest approach), time of flight and missile altitude. These are key items for evaluating missile performance. They are displayed on the terminal display.

2. The COMPUTE Message

The COMPUTE message really drives MISSILE. The following sequence of events occurs every time MISSILE is sent the COMPUTE message. COMPUTE requests target velocity and position data. Then COMPUTE sends this information, along with missile position and velocity to KINEMATICS. KINEMATICS replies with missile-to-target range, LOS rate, range-rate and time-to-go. LAUNCHER is then signaled to provide the range from the launch aircraft to the target. This information is then sent to the RF_SEEKER to determine which RF phase the missile is in and to determine the signal-to-noise ratio (SNR) of the signal the missile's radar receiver is receiving. COMPUTE then calculates A-pole, defined as the range from the launch aircraft to the target when the missile's radar enters the K-band acquisition mode (RF phases and modes are discussed in the RF_SEEKER section). If the missile is within range to use its IR seeker, IR_SEEKER is signaled to determine the IR phase and whether or not the radome has been ejected.

Next a series of computations and messages are executed resulting in the missile's current acceleration vector. First, COMPUTE gives KINEMATICS the missile's azimuth (az) and elevation (el) angles, and KINEMATICS returns the corresponding direction cosine matrix. Direction cosine matrices are used to transform vectors between different coordinate systems [Ref. 18]. In this case KINEMATICS returns the inertial to body coordinate system direction cosine matrix (TIB matrix, for transform inertial to body). This matrix is then multiplied by the missile-to-target range vector (which is in inertial coordinates) resulting in a missile-to-target range vector in missile body coordinates. This operation is also carried out on the other range vectors (i.e., second target and SOJs) if appropriate. The range vector in missile body coordinates is then sent to RF_SEEKER, which returns the seeker gimbal angle in missile body coordinates. COMPUTE then calculates total seeker angles (Psi and Theta) by summing the az and el angles with the seeker gimbal angles. KINEMATICS is sent these angles to use in computing the direction cosine matrix for the inertial to seeker coordinate system transformations (TIS).

The vector representing LOS rate in inertial coordinates is then transformed into seeker coordinates for use by GUIDANCE. COMPUTE also calculates missile altitude, and altitude rate which are used along with the LOS rate in seeker coordinates by GUIDANCE. GUIDANCE is sent these values, along with time-to-go and target position information and returns the guidance phase and commanded acceleration. Simply stated, given missile and target position and velocity information, GUIDANCE determines the required acceleration commands for the missile to intercept the target. COMPUTE then signals AUTOPILOT.COMPUTE with the commanded accelerations and receives back the achieved accelerations. KINEMATICS then, given the missile's altitude and velocity, returns the missile's mach number. COMPUTE signals THRUST with the missile's mach and altitude and receives assorted information such as missile mass, propulsion phase, and thrust. The achieved accelerations provided by AUTOPILOT.COMPUTE, along with missile mach and altitude, are sent to AIRFRAME.AERO which returns the missile's coefficient of drag and angle of attack. These values, along with missile position, velocity, and acceleration, are sent to KINEMATICS (equations of motion) which returns updated missile velocity information, pitch and dynamic pressure (Q). Finally the achieved accelerations provided by AUTOPILOT.COMPUTE are transformed from body to inertial coordinates. At this point COMPUTE has completed updating the missile's state.

C. KINEMATICS

The KINEMATICS, shown in Figure B.3, object calculates direction cosine matrices, missile acceleration, velocity, position and flight path data. KINEMATICS has the following messages: SETUP, INITIALIZE, MACH_NO, DIR_COS, COMPUTE, and EOM.

SETUP establishes the number of targets and the number of SOJs. INITIALIZE initializes ranges, range rates, and LOS rate. MACH_NO, given missile altitude and velocity, returns the missile's mach number. DIR_COS, given two reference angles between two different coordinate systems, returns the corresponding direction cosine matrix. This matrix is then used to transform vectors from one coordinate system to the other [Ref. 18].

The COMPUTE message calculates velocity, range, LOS rate and time-to-go. COMPUTE calculates missile-to-target velocity as the difference between target and missile velocity. The missile-to-target range vector is calculated as the difference between target position and missile position. Both a total range vector and unit range vector are also calculated. The range rate, defined as the rate of change of the missile-to-target range vector, is computed as the vector dot product of the range unit vector and the missile-to-target velocity vector. The LOS rate is then calculated as the vector cross product of the range unit vector and the missile-to-target velocity divided by total range. Range calculations are then performed for the SOJs if appropriate. Finally time-to-go, the estimated time to target intercept, is calculated. Time-to-go is computed, depending on various conditions, as either range to point of closest approach (miss distance) divided by range rate or target range divided by range rate. Care has to be taken to account for possible opening ranges immediately after missile launch , to avoid inaccurate miss distance calculations during initial flight phases, and to account for range rate becoming less than or equal to zero during the final phase of flight.

KINEMATICS' equations-of-motion message, EOM, calculates the missile's axial acceleration vector, pitch and heading. First pitch and heading are determined through trigonometric relations of angle-of-attack and velocity. Then a message is sent to ENVIRONMENT to get the air density. The air density, along with missile velocity, is used to compute the dynamic pressure (Q). Missile drag is then computed as the product of dynamic pressure, the missile's coefficient of drag and reference area (sref). Finally, the missile's axial acceleration vector is calculated and returned to the caller. This completes the description of KINEMATICS.

D. THE AIRFRAME

The AIRFRAME models the missile's aerodynamic characteristics and thrust characteristics. AIRFRAME's messages are INITIALIZE, AERO, and THRUST as shown in Figure B.4.

INITIALIZE initializes the propulsion phase, and various physical constants. The AERO message, given missile achieved acceleration, mach, altitude, dynamic pressure (Q), and the dome condition, returns the missile's coefficient of drag and angle-of-attack (AOA or Alpha). The AOA is the difference between the missile's velocity vector and its body vector. The drag coefficient is represented as a series of equations that are a function of missile mach. These equations represent a curve fit of data found through wind tunnel testing of the missile. In order for the missile's IR seeker to function, the radome, or dome, is ejected or blown off in the final phase of flight, increasing the drag coefficient.

THRUST uses missile mach, altitude, fuel mass, and missile mass to provide the thrust force and propulsion phase. The missile is modeled as having a solid fuel rocket motor. The rocket motor is fired at launch for a rail launch, or shortly after ejection for an eject launch. The initial propulsion phase is termed the boost phase. After the fuel is exhausted the missile enters the coast propulsion phase.

E. THE AUTOPILOT

AUTOPILOT accepts commanded accelerations and returns achieved accelerations dependent on the body responses of the missile. AUTOPILOT's messages are INITIALIZE, UPDATE_DIFF_EQS, COMPUTE, and ACCELERATIONS (see Figure B.5). INITIALIZE establishes initial values for autopilot constants. The autopilot has been modeled by differential equations which have been implemented as difference equations to avoid the instabilities caused by round-off error [Ref. 10]. UPDATE_DIFF_EQS updates the achieved accelerations through the difference equations. COMPUTE, given the commanded accelerations, returns the achieved accelerations, while ACCELERATIONS returns the instantaneous accelerations being experienced by the missile. AUTOPILOT serves as a good example of an object with a simple, well defined interface. This interface can be thought of as a standard interface in that, regardless of how the autopilot is modelled, this interface can remain unchanged. AUTOPILOT also serves as a good example of an object or a module that would be an excellent candidate to go into a library of missile subsystems. As production missiles mature, subsystems with new designs are incorporated. A library of various subsystems would allow the analyst to easily configure the simulation to match any production version of the missile. This would also permit the experimentation of new configurations with untested subsystem models.

F. GUIDANCE

The function of GUIDANCE is to guide the missile to the target. GUIDANCE withes the MATH object. Figure B.6 indicates GUIDANCE's messages are INITIALIZE and COMPUTE.

INITIALIZE, given missile altitude, target altitude, range, and velocity computes initial values for a number of guidance parameters. Guidance parameters such as guidance phase, horizontal target range, velocity, aspect angles and time estimates. COMPUTE, given time-to-go, missile altitude, altitude rate, velocity, axial acceleration, pitch,seeker gimbal angles, LOS rate, and target position and velocity, returns the guidance phase and acceleration commands. These commanded accelerations will guide the missile to intercept the target.

There are five guidance modes or phases: null commands,load bias, variable arc, altitude hold, and terminal. Immediately following launch the missile is in the null commands phase. The purpose of this phase is to ensure that no guidance commands are generated until the missile is safely clear of the launch aircraft. The load bias phase commands a five g pull-up maneuver until the missile achieves a 20 degree nose up attitude. The missile then enters the variable arc phase where it is commanded to climb to a predetermined altitude. Once reaching the predetermined altitude, the missile will remain at that altitude throughout the altitude hold phase. This enables the missile to dive on the target in the terminal phase, maximizing its available energy to provide maximum range. The missile uses proportional navigation with acceleration compensation to compensate for missile and target accelerations [Ref. 19]. Horizontal, or azimuth proportional navigation is used in all guidance phases except null commands, while vertical proportional navigation is used only in the terminal phase.

Messages internal to COMPUTE are GUIDANCE_MODE, ALTITUDE_HOLD_CMD, and GUIDANCE_COMMANDS. GUIDANCE_MODE replies with the current guidance phase. ALTITUDE_HOLD_COMMAND commands the missile to hold a constant altitude. GUIDANCE_COMMANDS calculates the commanded accelerations or guidance commands.

G. THE RF AND IR SEEKERS

The missile uses its RF or IR seeker to get information about the target [Ref. 20]. At longer ranges, the missile simply receives the RF energy reflected off the target from the launch aircraft's radar. This is known as the semi-active phase. The launch aircraft's radar operates in the X-band frequency range. Initially the aircraft's radar is in the X-band acquisition mode and upon acquiring the target enters the X-band track mode. At medium ranges, the missile's on-board radar activates to provide target information. This is known as the active phase. The missile's radar operates in the K-band frequency range. The missile's radar is initially in the K-band acquisition phase and enters the K-band track mode upon acquiring the target. At short ranges, the missile activates its IR seeker to acquire and track the target in the terminal phase of flight. Like the RF seeker, the IR seeker has an initial acquisition mode that is followed by a track mode once the target is acquired. In summary, at long ranges the launch aircraft's more powerful radar provides the best target information, while at medium ranges the missile's own radar provides the best target information, and at short ranges the missile's IR sensor provides the best target information. Modeling missile seekers is very complex and involves security issues. The seekers modeled here are rudimentary and future seeker work is discussed in Chapter VII.

The RF_SEEKER's external messages are SETUP, INITIALIZE, GIMBAL, and DETECTION (see Figure B.7). SETUP establishes the number of targets and SOJs, ECM techniques and power, and the target's RCS. INITIALIZE initializes the RF phase and radar power levels. GIMBAL, given the missile-to-target range vector, replies with the seeker gimbal angles. DETECTION, given missile-to-target range and launch aircraft-to-target range, responds with the RF phase, SNR, the selected target number, and bore sight error (BSE). The missile's BSE in this simulation is the same as the LOS, an imaginary line from the missile's seeker to the target. The BSE to target two is the angle between the BSE to target one and the LOS to target two. This parameter affects power levels received by the missile's radar.

RF_SEEKER's internal messages are BORESIGHT_ERROR , MSL_ANT_GAINS_SA, MSL_ANT_GAINS_A, SA_POWERS, A_POWERS, SA_PHASE, A_PHASE, SA_DETECT, and A_DETECT. BORESIGHT_ERROR calculates the bore sight error to the targets and SOJs. This defines the encounter geometry for the radar system and affects the corresponding power levels of the targets and SOJs seen by the radar system. MSL_ANT_GAINS_SA and

MSL_ANT_GAINS_A calculate the appropriate gain of the missile's radar antenna depending on the RF phase. SA_POWERS and A_POWERS calculates the power received by the missile's radar receiver based on the classic radar range equation [Ref. 21]. SA_MODE and A_MODE determine the RF mode, acquisition or track, for the semi-active and active RF phases. The SA_DETECT and A_DETECT determine whether or not the missile can detect a target given the power level received.

IR_SEEKER's messages are INITIALIZE and DETECTION as shown in Figure B.8. INITIALIZE establishes the IR phase and initializes search, acquisition and track times. DETECTION, given missile-to-target range, replies with the IR phase and Boolean RADOME_OFF, indicating the state of the radome.

H. THE LAUNCHER

The LAUNCHER object represents the aircraft that carries and launches the missile. LAUNCHER withes MATH for math operations and ENVIRONMENT for its SPEED_OF_SOUND message (see Figure B.9). LAUNCHER's messages are INITIALIZE, MSL_INIT, SETUP, LOG_DATA, GET_STATES, PUT_STATES, GET_DERIVATIVES, POLE and COMPUTE.

INITIALIZE initializes the launch aircraft's velocity and position. MSL_INIT provides launch aircraft initialization information for use by the missile. This message provides the missile with the launch aircraft's velocity and position, and certain radar characteristics based on the launch aircraft type. SETUP establishes the launch aircraft's guidance phase, lead angle, mach, and altitude. LOG_DATA provides the launch aircraft's position to the sender. GET_STATES returns the launch aircraft's state vector, while PUT_STATES updates the launch aircraft's state vector. The derivatives of the launch aircraft's state vector are supplied with GET_DERIVATIVES. POLE provides the distance from the launch aircraft to the target. The COMPUTE message provides the launch aircraft's velocity if the launch aircraft is in the pursuit guidance mode.

I. THE TARGETS

The TARGETS object models the aircraft that the missile is to intercept. The Ada package TARGETS is made up of four targets. Two of these targets, target one and target two, are treated as the primary targets, and targets three and four are treated as stand off jammers (SOJs). TARGETS withes MATH, ENVIRONMENT, and MISSILE, as shown if Figure B.10. TARGETS withes MATH for mathematical operations and uses ENVIRONMENT for its time keeping services and its SPEED_OF_SOUND message. TARGETS uses the MISSILE message MANEUVER_VALUE to coordinate target maneuvers.

TARGETS' messages are SETUP, INITIALIZE, LOG_DATA, GET_STATES, PUT_STATES, GET_DERIVATIVES, TGT_DATA, and COMPUTE. SETUP sets up the user entered target parameters. INITIALIZE computes the initial position and velocity parameters for the targets. INITIALIZE also calculates the final turn angle and weave period if appropriate. LOG_DATA provides the targets' position, velocity and heading data for data logging. GET_STATES and PUT_STATES put and get the targets' state vector (position), respectively. GET_DERIVATIVES provides the derivatives of the targets' state vector (velocity). TGT_DATA provides both target position and velocity data to the sender. TGT_ASPECT provides the sender with the target one's aspect angle.

COMPUTE is the heart of TARGETS. COMPUTE calculates the targets' position, velocity and heading angle dependent on target maneuver. Appropriate conditions are checked resulting in the setting of flags and the times for the corresponding target maneuver. Then the build-up time must be considered. The build-up time is the time from the initiation of the maneuver until the desired number of g's is achieved. This models the real world condition that commanded maneuvers are not achieved instantaneously. The rate of change of the target's heading is then calculated along with the number of g's the target is experiencing. The current target heading angle is then compared with the final desired turn angle. The target velocity vector and mach are then computed. Finally, the second target's position is computed based on its geometric relation with the first target. To summarize, the target heading angle is calculated based on maneuver type. Because the targets have a constant velocity, the target heading angle is required to compute the velocity vector, which in turn permits the computation of the targets' position vector.

J. THE ENVIRONMENT

The ENVIRONMENT provides atmospheric and time information. ENVIRONMENT's messages are SET_TIME, TIME, AIR_DENSITY, and SPEED_OF_SOUND (see Figure B.11). ENVIRONMENT is the simulation's time keeper. SET_TIME allows the system time to initialized and incremented. The TIME message provides the current system time. AIR_DENSITY provides the air density for a given altitude, while SPEED_OF_SOUND provides the speed of sound at a given altitude [Ref. 9].

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis has explored using object oriented techniques and software engineering principles in conjunction with the Ada programming language to develop a missile flight simulation. By using these techniques and principles the problem space is accurately mapped into software. This, along with the principles of abstraction, information hiding, modularity, loose coupling, and strong cohesion produced a simulation that is easily understood, modifiable, efficient and reliable.

Although understandability can be very subjective, all of the missile analysts who reviewed the simulation agreed that the code is much more easily understood than previous FORTRAN versions. Modularity, high cohesion, and loose coupling permitted the simulation to be modified in easily. Modules were designed to serve a single purpose and to make use of only the data or control information presented by the interfaces of other modules. All the interfaces are well defined and are standard for that particular module. A good example is the abstraction of the missile airframe subsystem. By being modular and having a standard well defined interface, this subsystem evolved from a program stub to a fairly complex model with minimal programming effort. Also by having a standard well defined interface between objects or modules, a library of different models can be built to explore different missile and target configurations. The simulation is simply relinked with the desired module. This allows a number of different models to be built relatively quickly. These models then can be used for comparison studies.

Through abstraction, information hiding, and modularity a very efficient user interface was developed. The simulation has also proven itself to be highly reliable, producing consistent results that agree with missile system expert's predictions. The simulation has also proven to be quite robust, surviving the most mischievous users without crashing.

B. RECOMMENDATIONS AND FUTURE WORK

This thesis has laid the ground work for a generic missile flight simulation that will evolve to model existing classified missile systems. The continued improvement of the various models used in the simulation is highly recommended. Future work includes consulting local Pacific Missile Test Center missile system experts to improve and validate models. This will involve modifying the generic MISSILE to model a specific missile system. Work could then progress on the classified aspects of the missile's radar system and the electronic countermeasures aspects of the simulation. This might include modifying the ENVIRONMENT to model IR environmental concerns such as fog and haze, and radar environmental concerns such as ground clutter. The objects or modules will permit the experts to concentrate on the detailed level of their particular area of expertise, isolated, if desired, from the programming details of the other objects. Drivers will be developed to permit these objects to be tested stand-alone as individual components.

Work is also underway on the SYSTEM_SPECIFIC and LOW_LEVEL objects for the Apple MacIntosh computer. Once these objects are developed the simulation will run on the MacIntosh computer. A particularly good area for future work involves the data the simulation generates. The simulation is capable of generating large amounts of data that must be interpreted by analysts. Work is progressing on a object or package that will plot the data sets. Future work might include an expert system that reviews the data to assist the analyst in identifying problem areas in missile performance.

APPENDIX A

CONTROL AND SUPPORT OBJECT DIAGRAMS

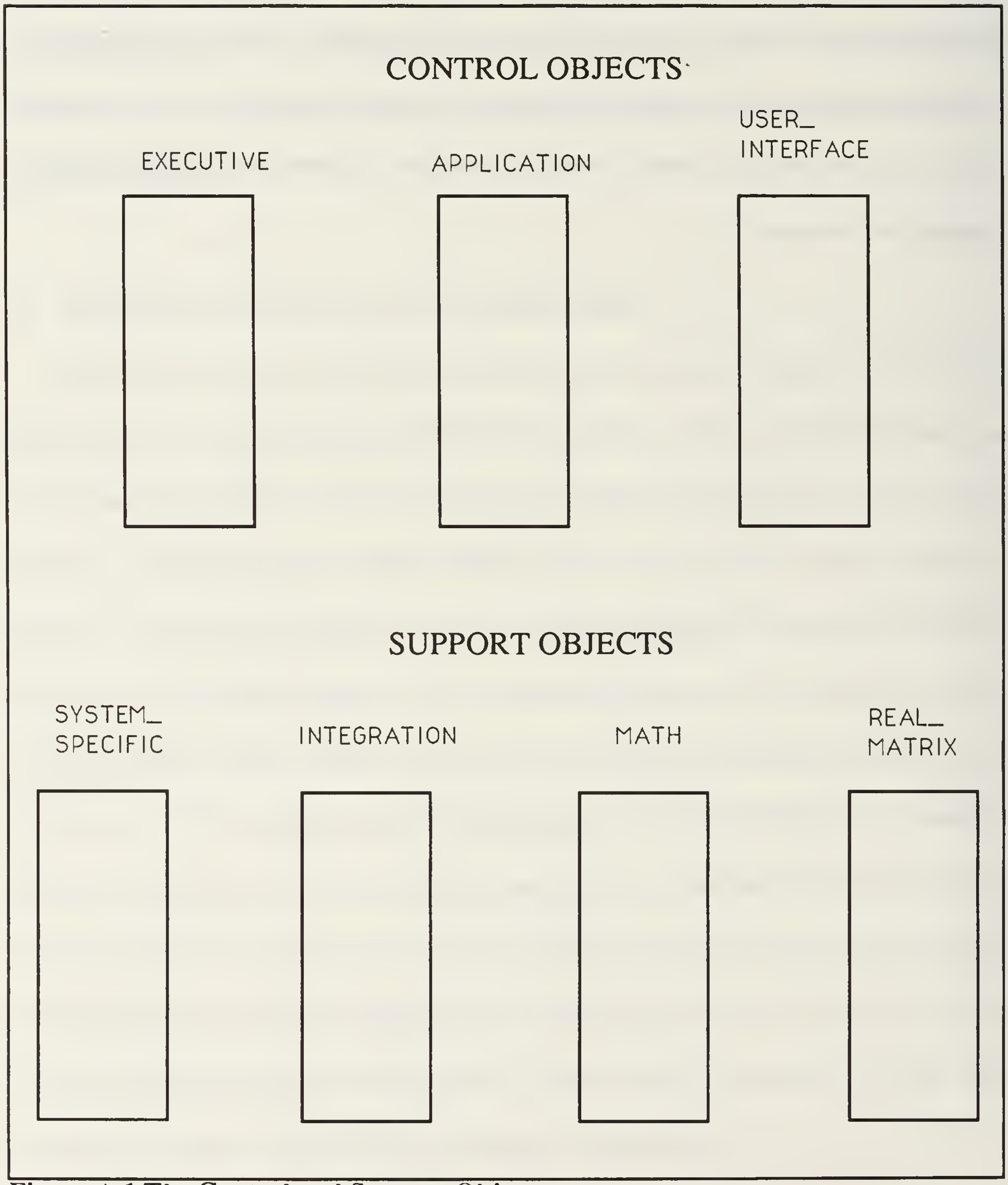


Figure A.1 The Control and Support Objects

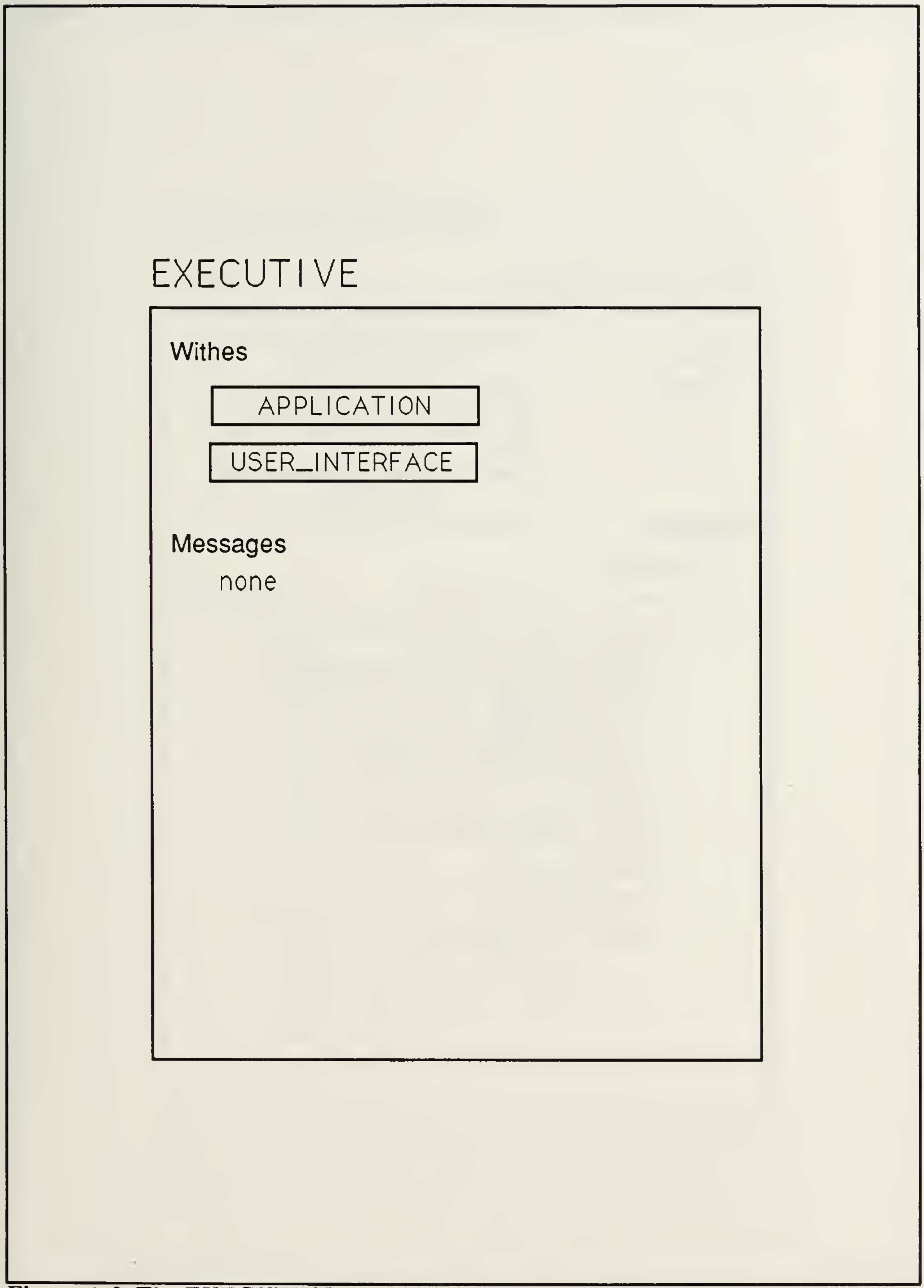


Figure A.2 The EXECUTIVE

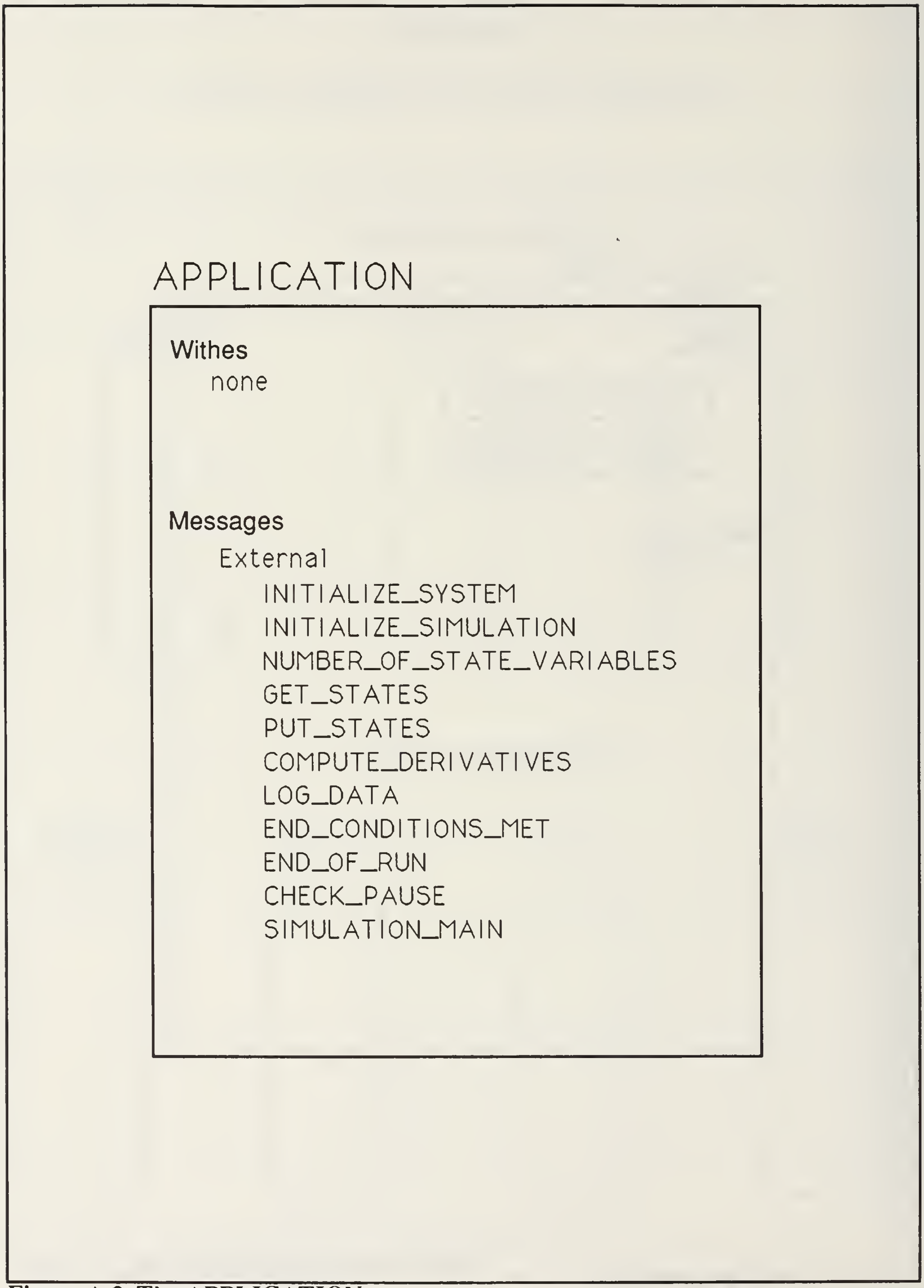


Figure A.3 The APPLICATION

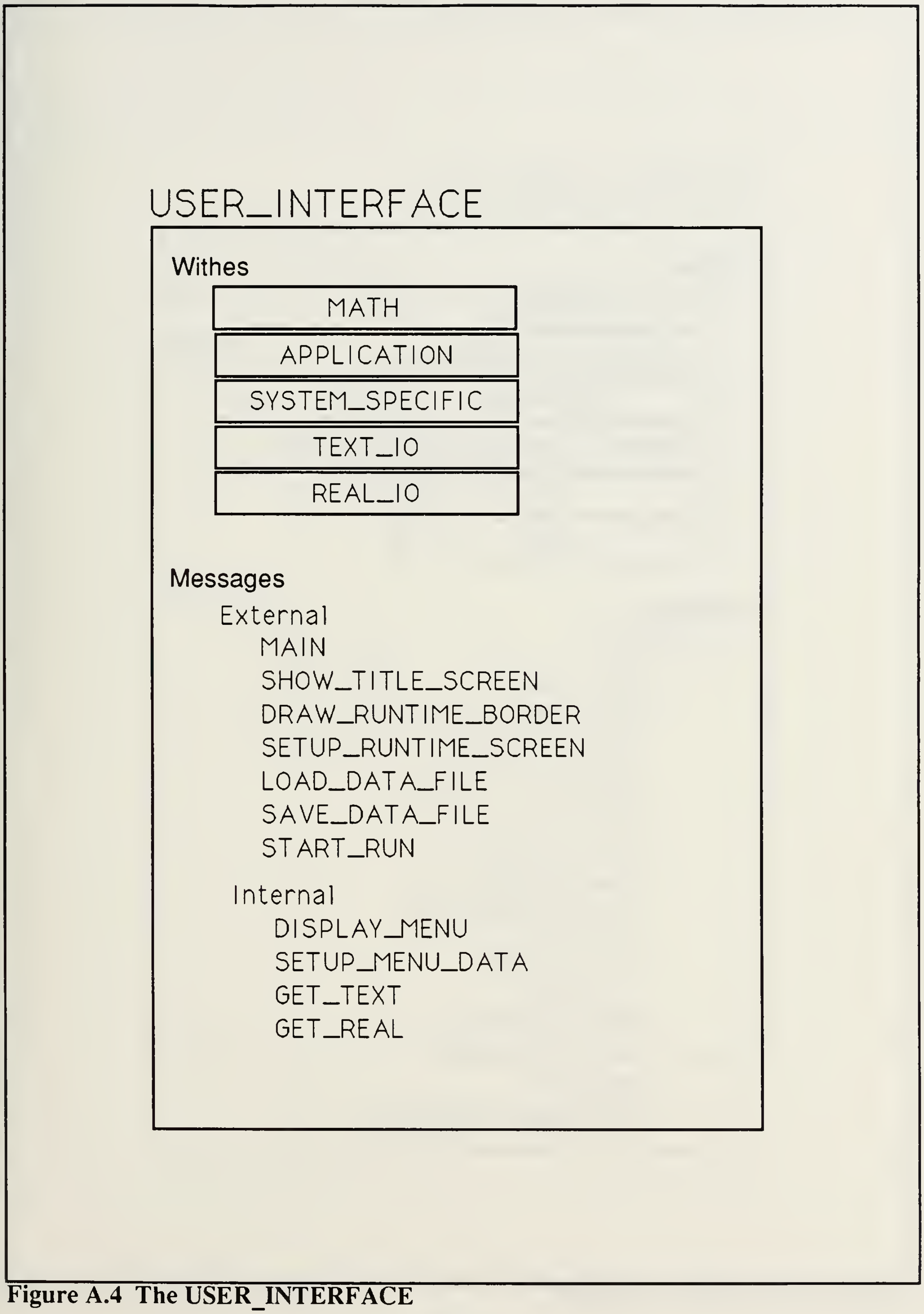


Figure A.4 The USER_INTERFACE

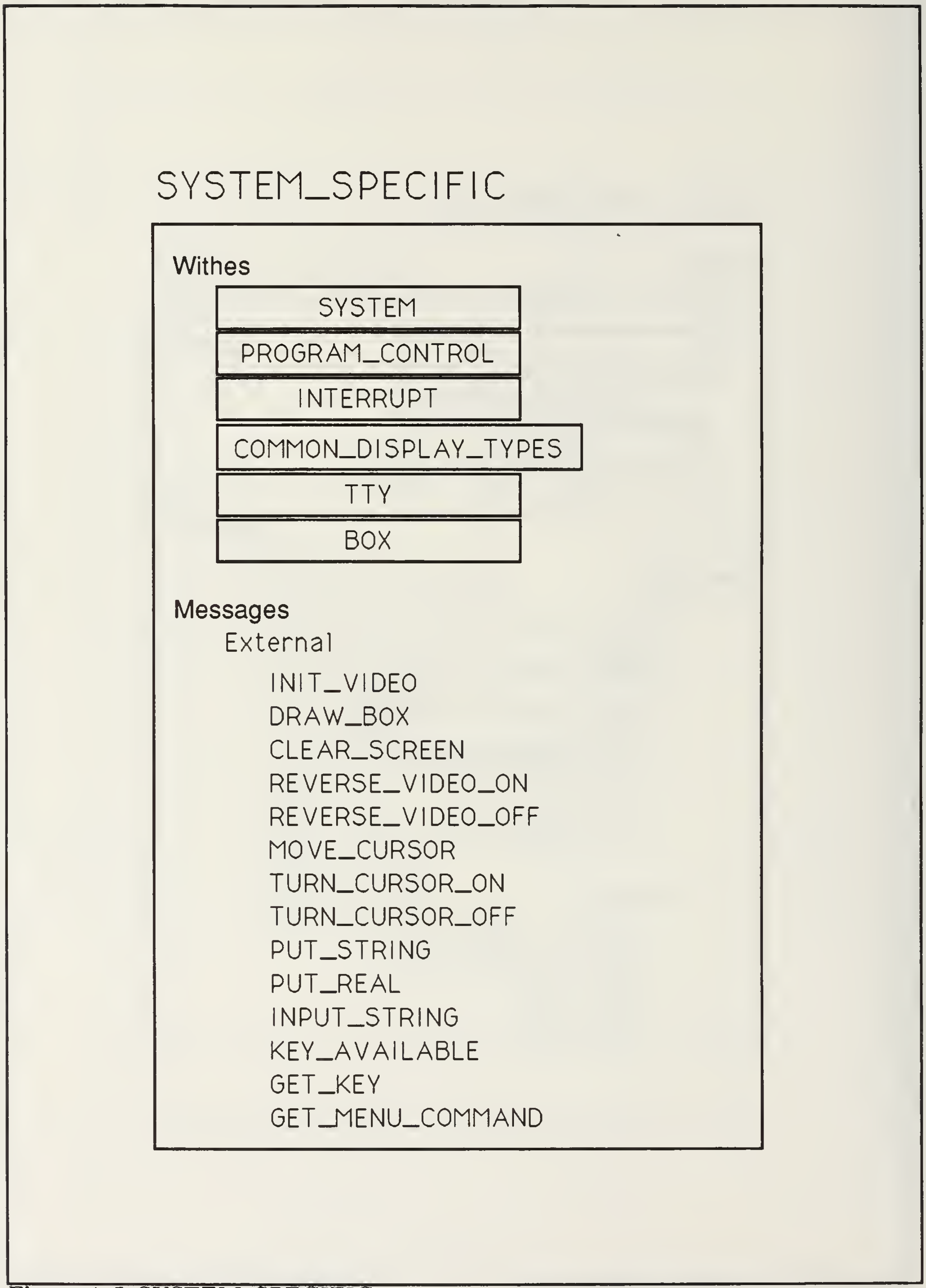


Figure A.5 SYSTEM_SPECIFIC

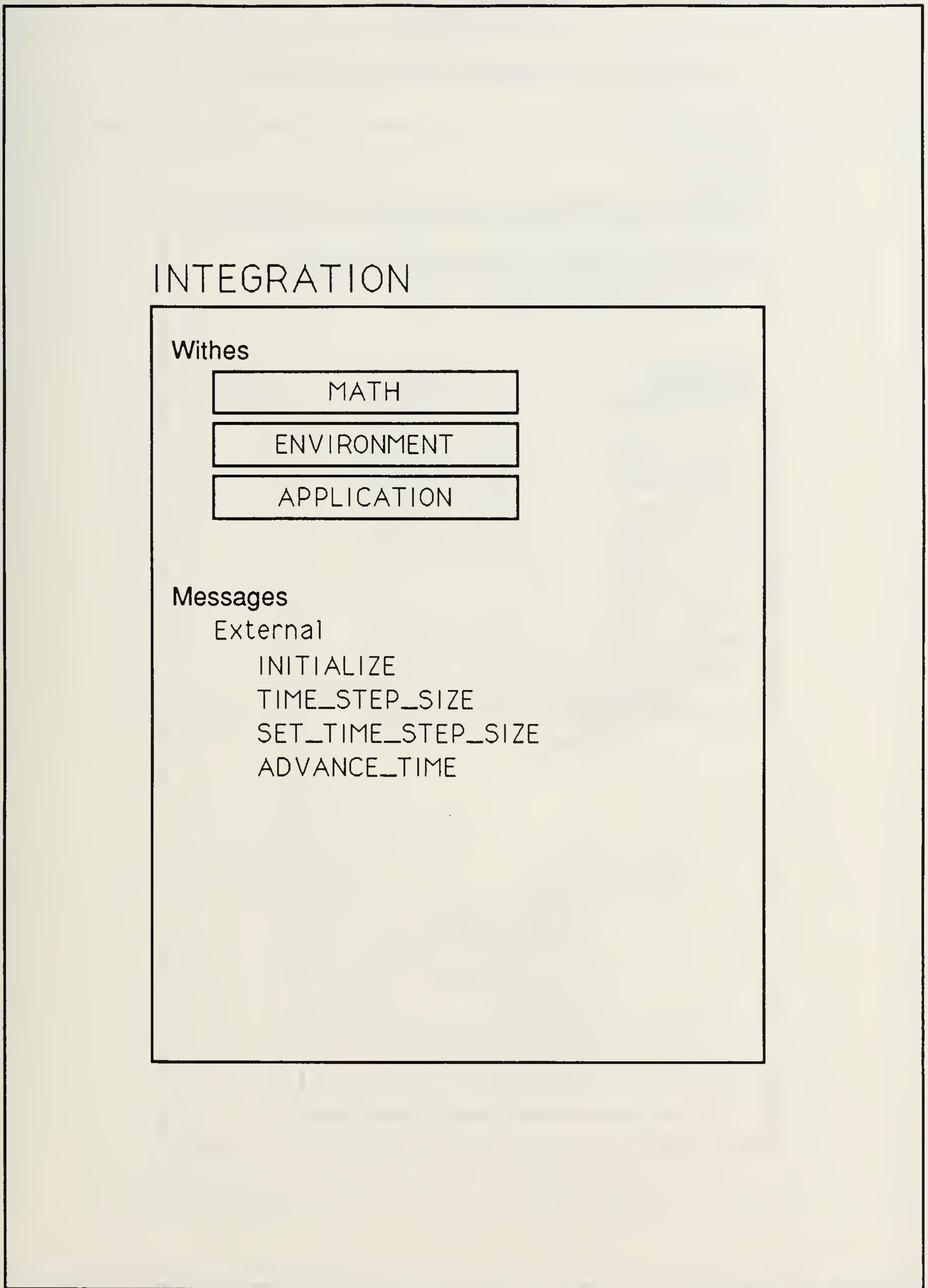


Figure A.6 INTEGRATION

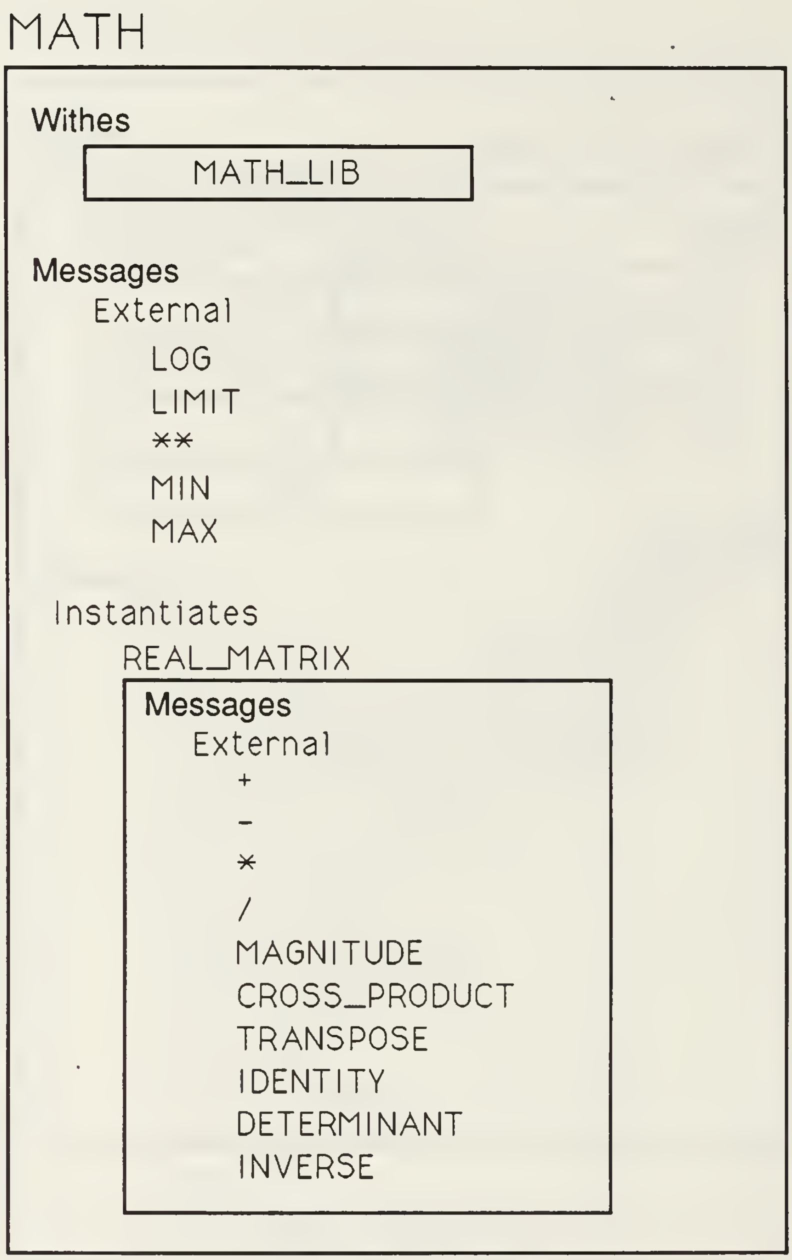


Figure A.7 MATH

APPENDIX B

PROBLEM SPACE OBJECT DIAGRAMS

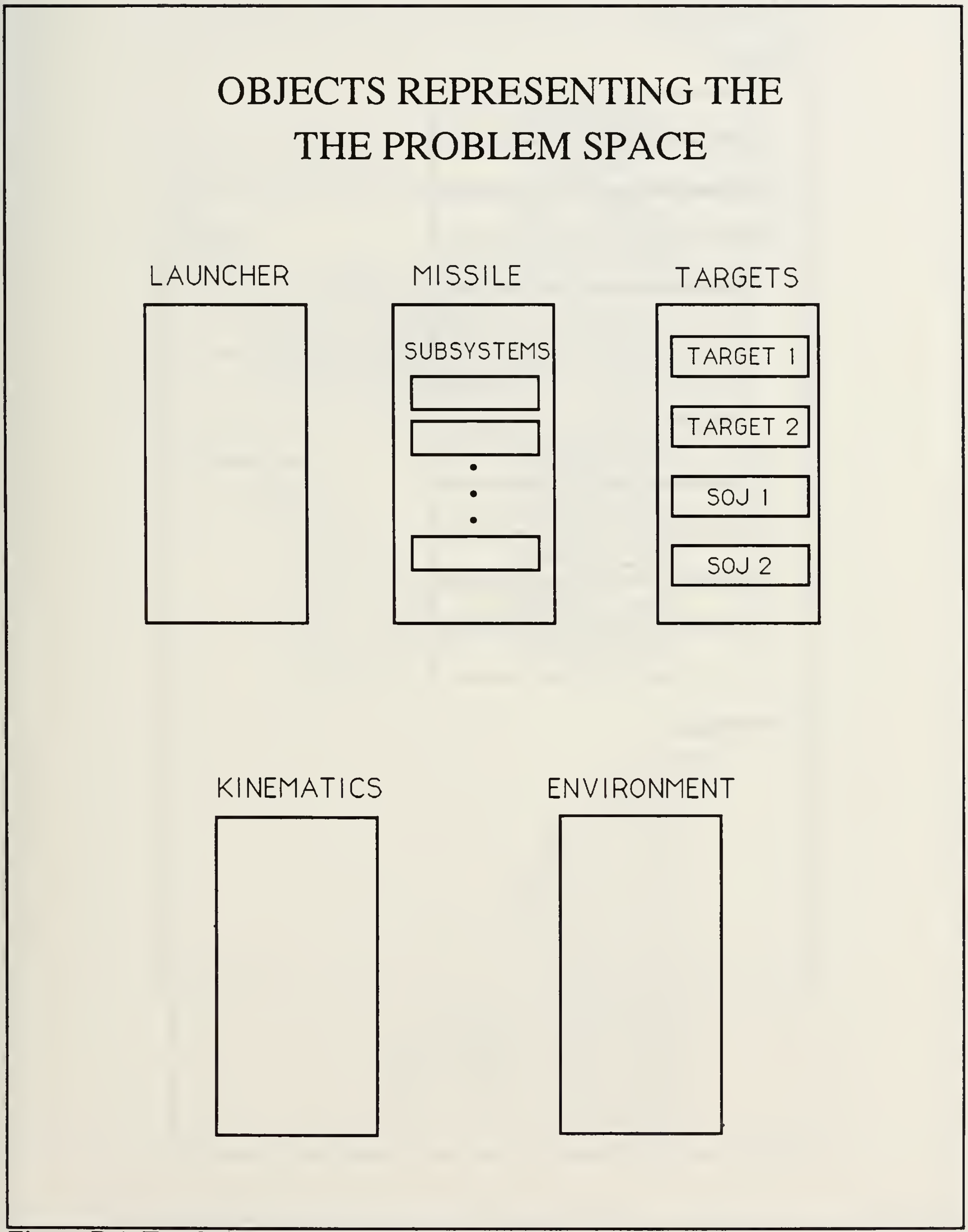


Figure B.1 The Software Mapping of the Problem Space

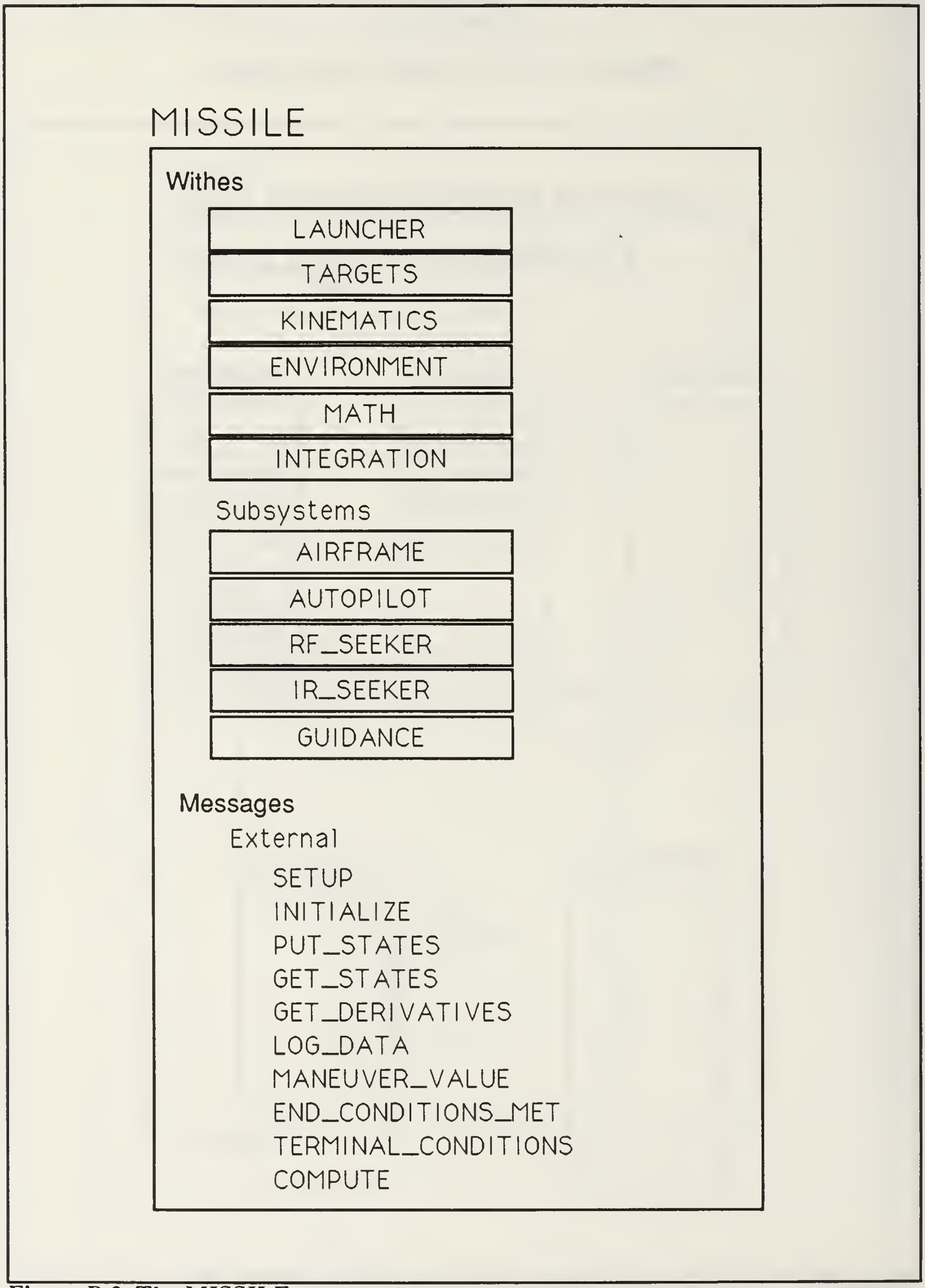


Figure B.2 The MISSILE

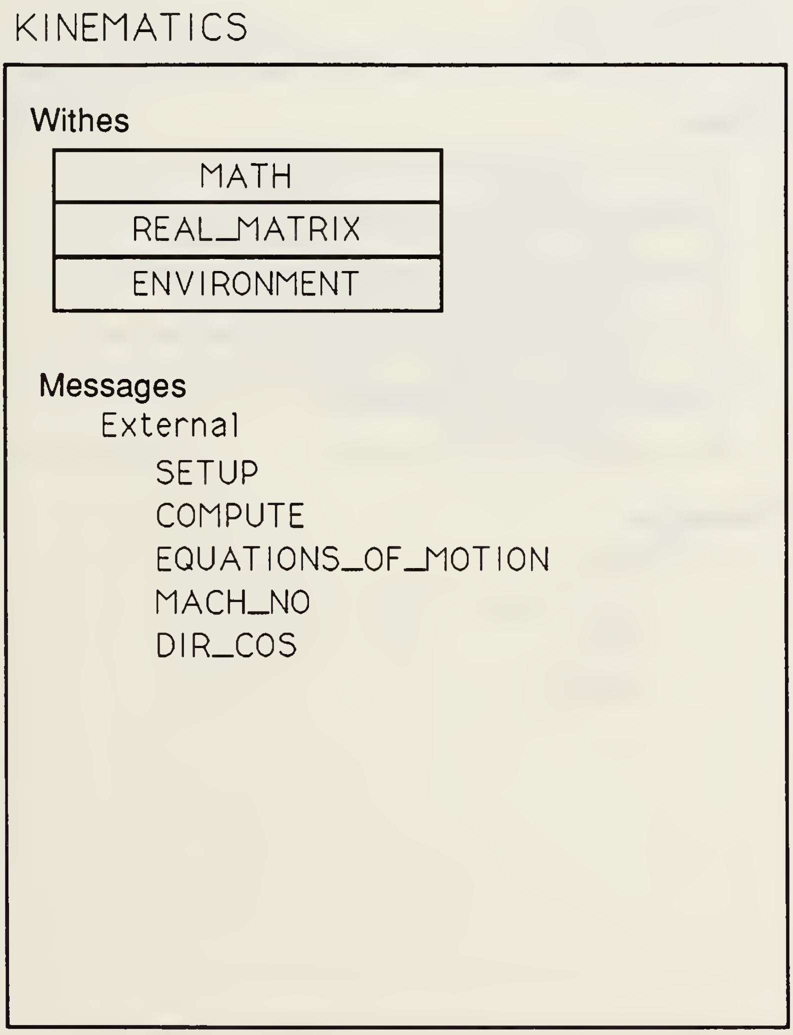


Figure B.3 KINEMATICS

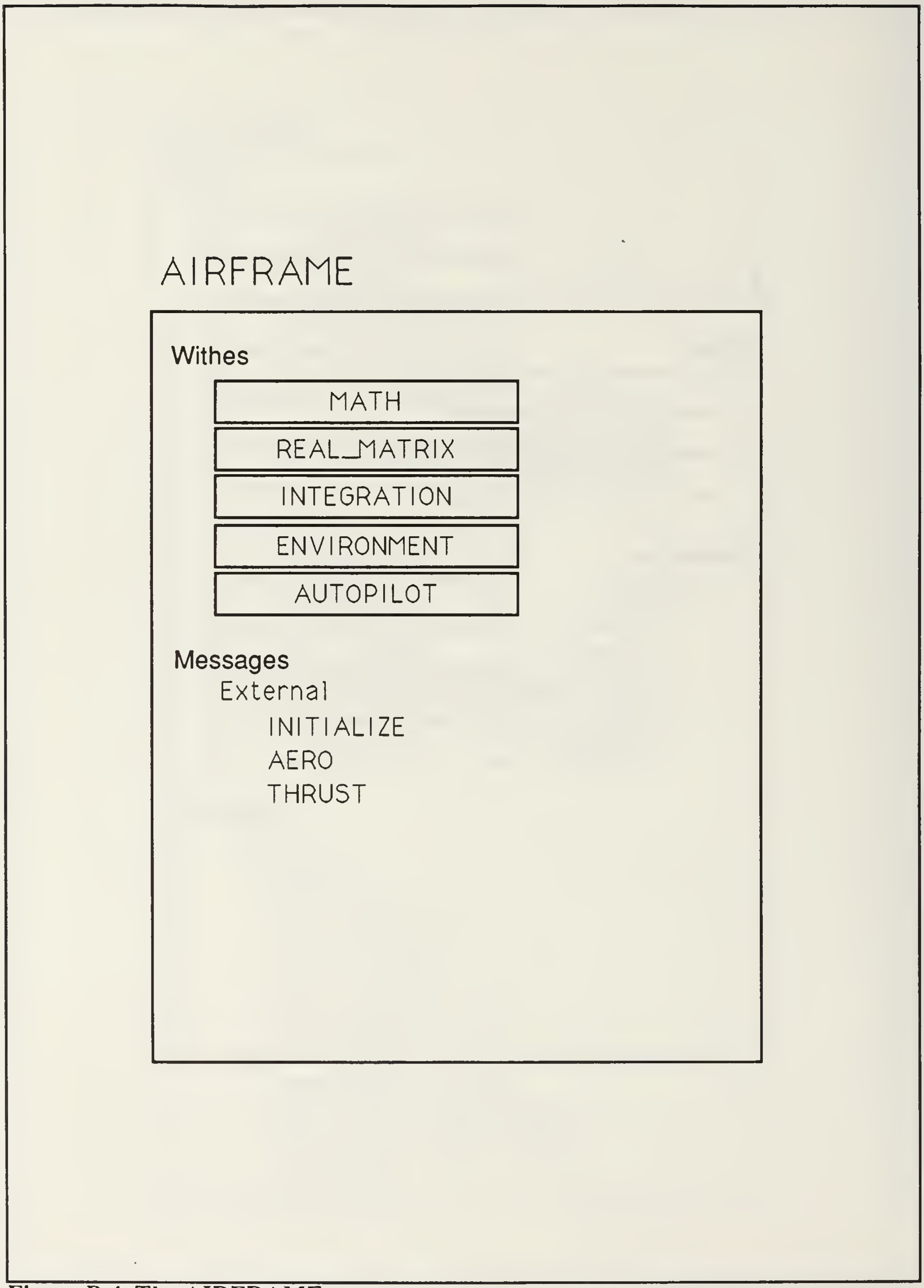


Figure B.4 The AIRFRAME

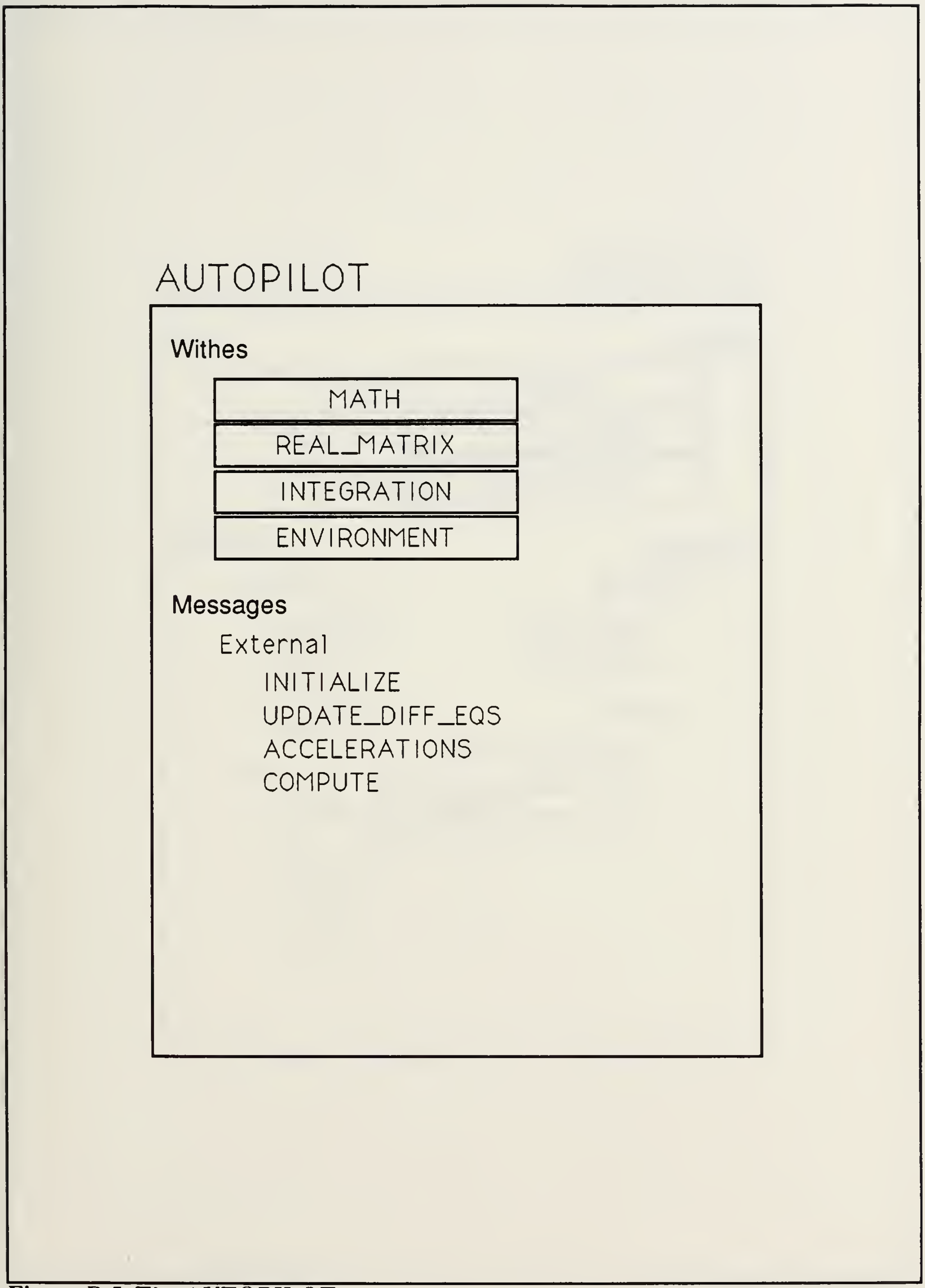


Figure B.5 The AUTOPILOT

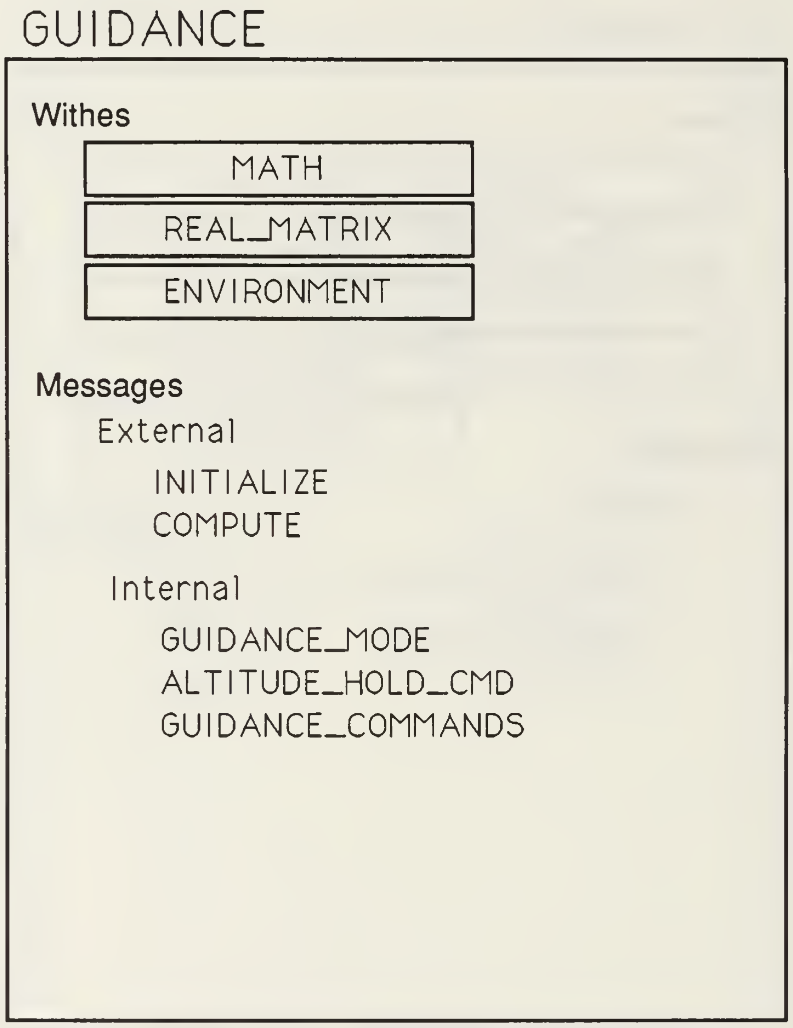


Figure B.6 GUIDANCE

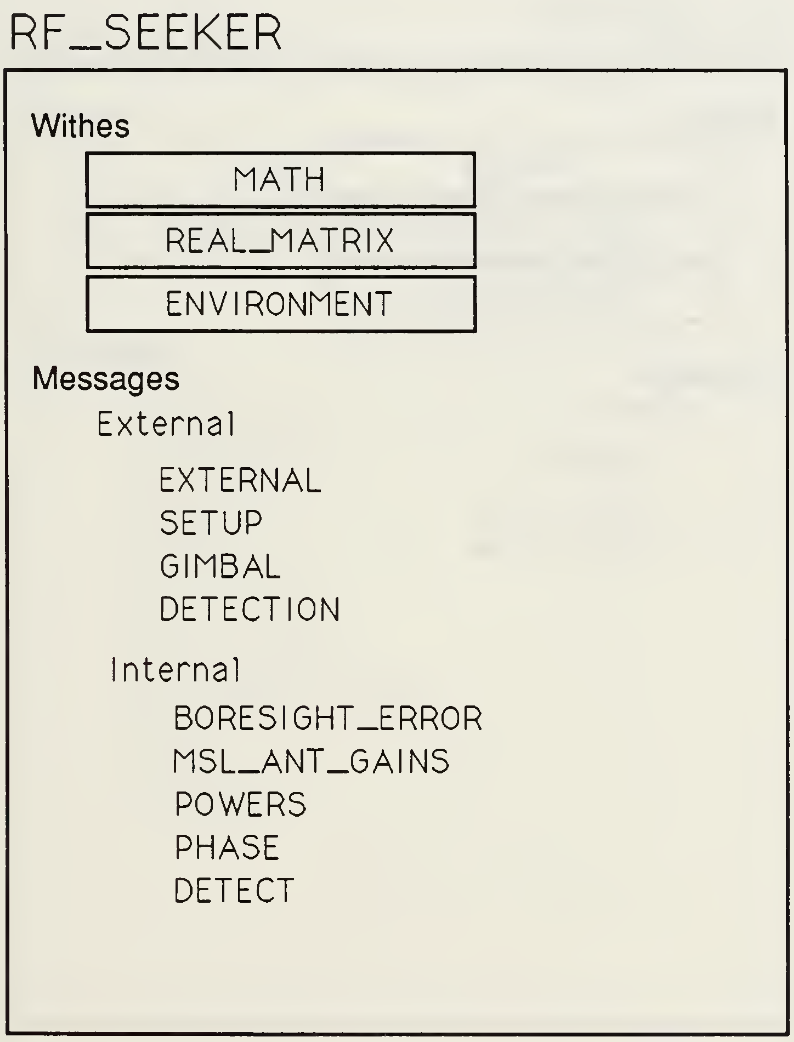


Figure B.7 The RF_SEEKER

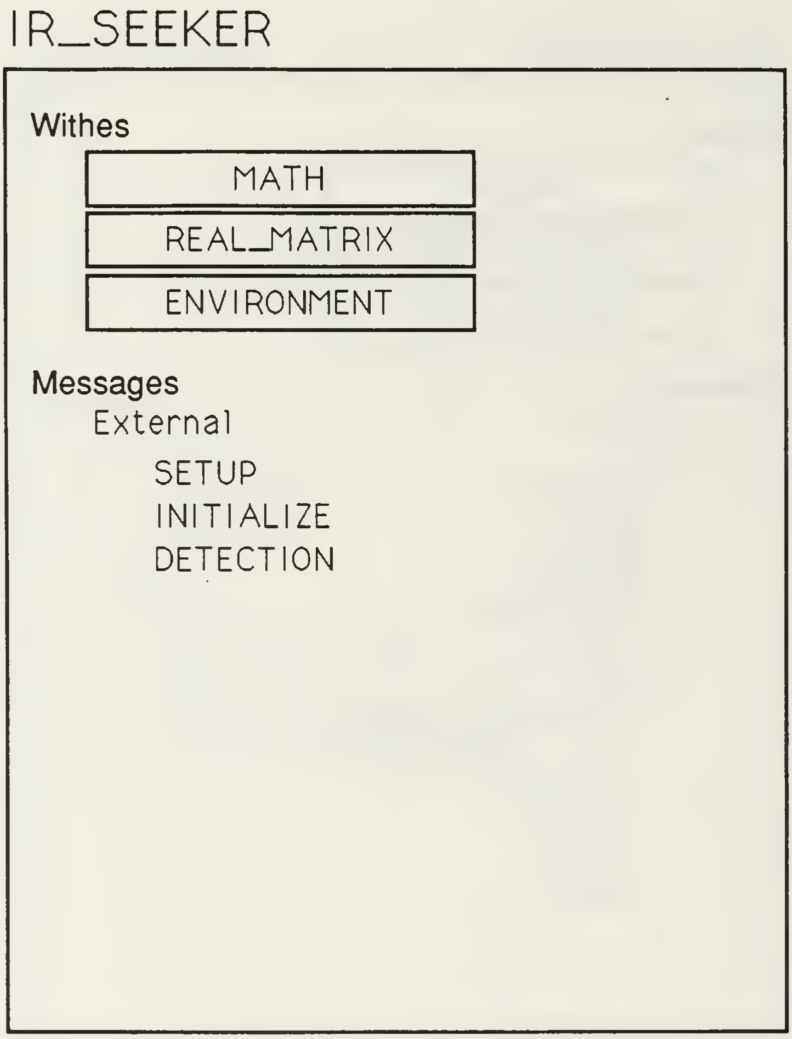


Figure B.8 The IR_SEEKER

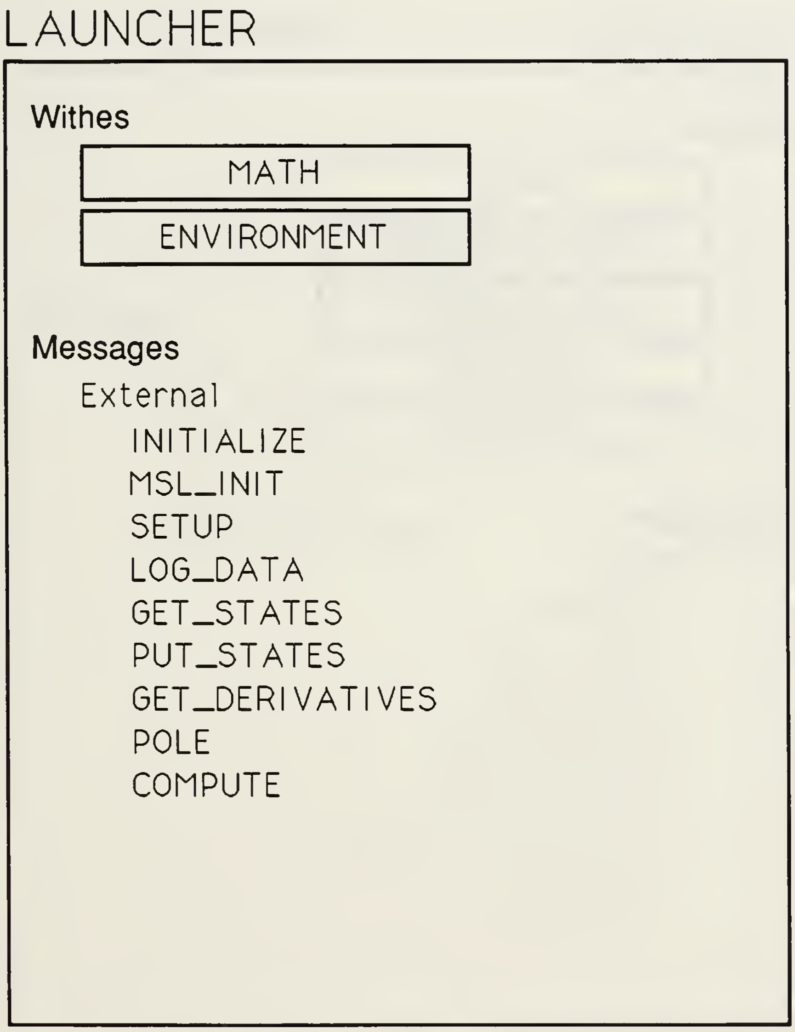


Figure B.9 The LAUNCHER

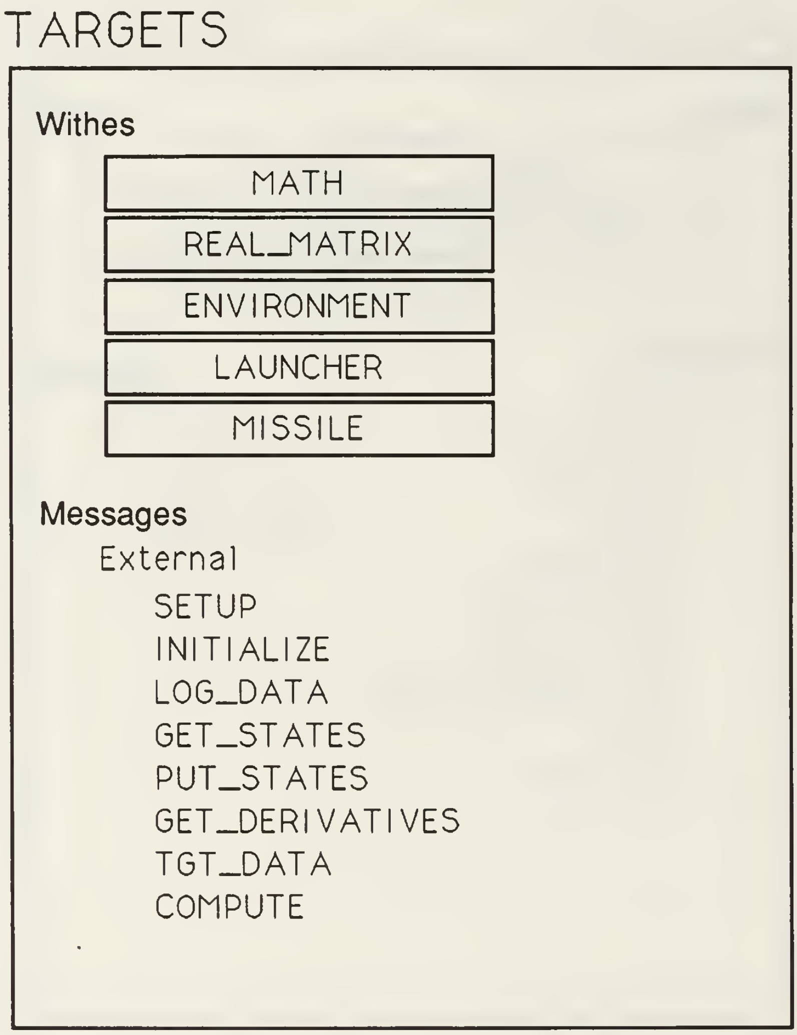


Figure B.10 The TARGETS

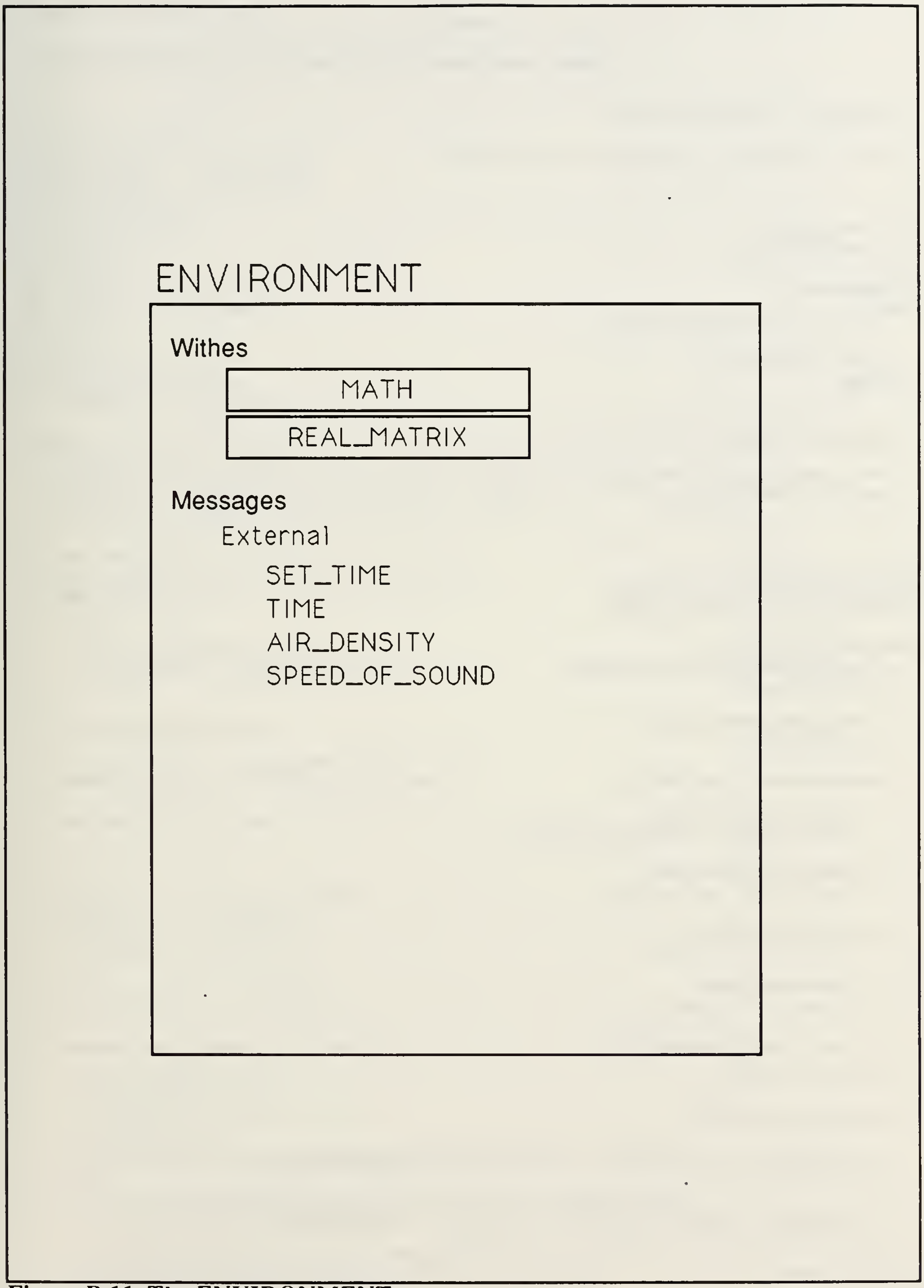


Figure B.11 The ENVIRONMENT

APPENDIX C

CONTROL OBJECT SOURCE CODE LISTING

```
-------------------------------------------------------------------------------
-- Simulation Executive Procedure
--
-- This is the top level procedure for the simulation.
--
-------------------------------------------------------------------------------

with APPLICATION;
with USER_INTERFACE;

procedure SIM is
begin
    APPLICATION.INITIALIZE_SYSTEM;

    USER_INTERFACE.MAIN;
end SIM;

-------------------------------------------------------------------------------
-- Application Package Specification
--
-- This package contains the application specific functions called by the
-- general Executive Package.
--
-------------------------------------------------------------------------------

with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;
with MODEL_TYPES; use MODEL_TYPES;

package APPLICATION is
    function NUMBER_OF_STATE_VARIABLES
    return integer;

    function END_CONDITION_MET
    return boolean;

    procedure INITIALIZE_SYSTEM;

    procedure SIMULATION_MAIN(
        SETUP_VALUES : in SETUP_VALUES_TYPE);

    procedure GET_DERIVATIVES (
        DERIVATIVES : out VECTOR);

    procedure GET_STATES (
        STATES : out VECTOR);

    procedure PUT_STATES (
        STATES : in VECTOR);

    procedure COMPUTE_DERIVATIVES;

    procedure LOG_DATA;

    procedure END_OF_RUN;

    procedure GET_TIMES(TIME_ARRAY : out VECTOR; NUMBER_OF_VALUES : out
        integer);

    procedure GET_VARIABLE(WHICH_VARIABLE : in integer; RETURN_VARIABLE :
        out VECTOR; NUMBER_OF_VALUES : out integer);

end APPLICATION;
```

```
--------------------------------------------------------------------------------
-- Application Package Body
--
-- This package contains the application specific functions called by the
-- general Executive Package.
--
--------------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;
with ENVIRONMENT;
with SYSTEM_SPECIFIC;
with USER_INTERFACE;
with MISSILE;
with LAUNCHER;
with TARGETS;
with Calendar; use Calendar;
with Text_io;
with REAL_IO;
with INTEGRATION;
with LOW_LEVEL_GRAPHICS;
with UNCHECKED_DEALLOCATION;

package body APPLICATION is
    LOGGING_TO_DISK : boolean;
    MEMORY_FULL : boolean;
    LOG_INTERVAL : integer; -- Frames between data logging
    LOG_RECORD : LOG_RECORD_TYPE;

    type LOGGED_DATA;
    type LOGGED_DATA_PTR_TYPE is access LOGGED_DATA;
    type LOGGED_DATA is
    record
         time : real;
        DATA : LOG_RECORD_TYPE;
        NEXT_RECORD : LOGGED_DATA_PTR_TYPE;
    end record;

    pragma PACK(LOGGED_DATA);

    FIRST_DATA_RECORD : LOGGED_DATA_PTR_TYPE := null;
    CURRENT_DATA_RECORD : LOGGED_DATA_PTR_TYPE;

    TERMINATE_FLAG : boolean;

    START_TIME  : Time;

    LOG_DATA_FILE : Text_io.File_type;
    MISSILE_STATES : VECTOR(1..6);
    LAUNCHER_STATES : VECTOR(1..3);
    TARGET_STATES : VECTOR(1..3);

    procedure LOG_DATA is separate;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SHOW_TERMINAL_CONDITIONS is
        A_POLE : REAL;
        MISS_DISTANCE: REAL;
        TIME_OF_FLIGHT: REAL;
        ALTITUDE: REAL;
        RDOT: REAL;
        STOP_CONDITION: STOP_CONDITION_TYPE;
    begin
        MISSILE.TERMINAL_CONDITIONS(A_POLE, MISS_DISTANCE, TIME_OF_FLIGHT,
            ALTITUDE,RDOT,STOP_CONDITION);

        SYSTEM_SPECIFIC.PUT_REAL( 2,29, TIME_OF_FLIGHT);
        SYSTEM_SPECIFIC.PUT_REAL( 3,67,ALTITUDE, 1); -- Altitude
        SYSTEM_SPECIFIC.PUT_REAL( 15,67,MISS_DISTANCE/ FEET_PER_NMI , 1);
        SYSTEM_SPECIFIC.PUT_REAL(16,67,RDOT, 1);
        SYSTEM_SPECIFIC.PUT_STRING(21,41, "Reason for termination   :");
        SYSTEM_SPECIFIC.PUT_STRING(21,68, STOP_REASON(STOP_CONDITION));

        if MISS_DISTANCE < 10_000.0 then
            SYSTEM_SPECIFIC.PUT_STRING(22,41, "Miss Distance, ft        :");
            SYSTEM_SPECIFIC.PUT_REAL(22,67,MISS_DISTANCE,2); -- Miss-Dist, ft
        end if;

        if MISSILE.IS_ACTIVE then
            SYSTEM_SPECIFIC.PUT_STRING(23,41, "A-Pole, NMI              :");
            SYSTEM_SPECIFIC.PUT_REAL(23,67, A_POLE / FEET_PER_NMI); -- A-Pole
        end if;
    end SHOW_TERMINAL_CONDITIONS;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure INITIALIZE_SIMULATION(OUTPUT_FILE : in string) is
        procedure FREE is new UNCHECKED_DEALLOCATION(
            LOGGED_DATA, LOGGED_DATA_PTR_TYPE);

        PREV_DATA_RECORD : LOGGED_DATA_PTR_TYPE;
    begin
        if FIRST_DATA_RECORD /= null then
            CURRENT_DATA_RECORD := FIRST_DATA_RECORD;
            while CURRENT_DATA_RECORD.NEXT_RECORD /= null loop
                PREV_DATA_RECORD := CURRENT_DATA_RECORD;
                CURRENT_DATA_RECORD := CURRENT_DATA_RECORD.NEXT_RECORD;
                FREE(PREV_DATA_RECORD);
            end loop;
            FREE(CURRENT_DATA_RECORD);
            FIRST_DATA_RECORD := null;
        end if;

        TERMINATE_FLAG := false;
        LAUNCHER.INITIALIZE;
        TARGETS.INITIALIZE;
        MISSILE.INITIALIZE;
        USER_INTERFACE.SETUP_RUNTIME_SCREEN;
        START_TIME := Clock;
        MEMORY_FULL := false;
        if LOGGING_TO_DISK then
            Text_io.Create(LOG_DATA_FILE, Text_io.Out_file, OUTPUT_FILE);
            Text_io.Put(LOG_DATA_FILE ,
                "Time      ,Msl_Pos_N ,Msl_Pos_E ,Altitude  ,R_LAC_Tgt ,");
            Text_io.Put(LOG_DATA_FILE ,
                "Velocity  ,Msl_Head_A,Pitch_Ang ,Msl_Mach  ,Range_Tgt ,");
            Text_io.Put(LOG_DATA_FILE ,
                "Range_Rate,Time_To_Go,Skr_TotAng,Skr_El_Ang,Skr_Az_Ang,");
            Text_io.Put(LOG_DATA_FILE ,
                "El_Rate_T1,Az_Rate_T1,Alt_Rate  ,Az_Acc_Cmd,El_Acc_Cmd,");
            Text_io.Put(LOG_DATA_FILE ,
                "X_Axis_Acc,Az_Acc_Ach,El_Acc_Ach,Msl_Mass  ,Thrust    ,");
            Text_io.Put(LOG_DATA_FILE ,
                "Tot_Alpha ,Alpha     ,Beta      ,SNR       ,Drag_Coef ,");
            Text_io.Put(LOG_DATA_FILE ,
                "Tgt Type  ,Tgt 1 BSE ,Tgt 2 BSE ,Tgt 3 BSE ,Tgt 4 BSE ,");
            Text_io.Put(LOG_DATA_FILE ,
                "LAC_Pos_N ,LAC_Pos_E ,LAC_Alt   ,T1_Pos_N  ,T1_Pos_E  ,");
            Text_io.Put(LOG_DATA_FILE ,
                "T1_Alt    ,T2_Pos_N  ,T2_Pos_E  ,T2_Alt    ,T1_Vel    ,");
            Text_io.Put(LOG_DATA_FILE ,
                "T1_Mach   ,T1_Hdg_Ang,Guid_Phase,Msl_Phase ,RF_Phase  ,");
            Text_io.Put(LOG_DATA_FILE ,
                "IR_Phase  ,Radom_Off ,Prop_Phase");
            Text_io.New_line(LOG_DATA_FILE);
        end if;

        COMPUTE_DERIVATIVES;
    end INITIALIZE_SIMULATION;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure CHECK_PAUSE is
        KEY_AVAILABLE : boolean;
        KEY : character;
    begin
        USER_INTERFACE.KEYBOARD_HANDLER.KEY_AVAILABLE(KEY_AVAILABLE);
        if not END_CONDITION_MET and not KEY_AVAILABLE then
            return;
        end if;

        USER_INTERFACE.KEYBOARD_HANDLER.GET_KEY_NOWAIT(KEY, KEY_AVAILABLE);
        if END_CONDITION_MET or else (KEY_AVAILABLE and KEY = ' ') then
            SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;

            if END_CONDITION_MET then
                SYSTEM_SPECIFIC.PUT_STRING(24, 12,
                    " Run completed. Press space bar to return to main menu ");
            else
                SYSTEM_SPECIFIC.PUT_STRING(24, 15,
                    " Press space bar to continue run, T to terminate ");
            end if;

            SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;

            loop
                USER_INTERFACE.KEYBOARD_HANDLER.GET_KEY_WAIT(KEY);

                if KEY = ' ' then
                    if not END_CONDITION_MET then
                        USER_INTERFACE.DRAW_RUNTIME_BORDER;
                    end if;
                    exit;
                end if;

                if KEY = 't' or KEY = 'T' then
                    TERMINATE_FLAG := true;
                    exit;
                end if;
            end loop;
        end if;
    end CHECK_PAUSE;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure SIMULATION_MAIN(SETUP_VALUES : in SETUP_VALUES_TYPE) is
        FRAME_NUMBER   : integer;
        NEXT_LOGGING_FRAME : integer;
    begin
        LOGGING_TO_DISK := boolean'val(YES_NO_TYPE'pos(SETUP_VALUES.LOG_DATA));

        LAUNCHER.SETUP(SETUP_VALUES.INT_GUIDANCE,
            DEG_TO_RAD*SETUP_VALUES.INT_LEAD_ANGLE_IC,
            SETUP_VALUES.INT_MACH_IC,
            1000.0*SETUP_VALUES.INT_ALTITUDE_IC,
            SETUP_VALUES.INT_TYPE_IC);

        TARGETS.SETUP(DEG_TO_RAD*SETUP_VALUES.TGT_ASPECT_IC,
            SETUP_VALUES.TGT2_ANGLE_IC,
            SETUP_VALUES.TGT_MACH_IC,
            SETUP_VALUES.TGT_TURN_G_IC,
            SETUP_VALUES.TGT_WEAVE_PER_IC,
            SETUP_VALUES.TGT_TURNON_VALUE_IC,
            DEG_TO_RAD*SETUP_VALUES.TGT_TURN_ANG_IC,
            SETUP_VALUES.TGT_BUILDUP_TIME_IC,
            SETUP_VALUES.TGT_RANGE_IC,
            SETUP_VALUES.TGT_ALTITUDE_IC,
            SETUP_VALUES.SOJ_ANGLE_IC,
            SETUP_VALUES.MANEUVER_KIND,
            SETUP_VALUES.TGT_TWO_IC,
            SETUP_VALUES.SOJ_ONE_IC,
            SETUP_VALUES.SOJ_TWO_IC,
            SETUP_VALUES.TURN_ON_PARAMETER);

        MISSILE.SETUP(SETUP_VALUES.LAUNCH_TYPE,
            SETUP_VALUES.TGT_RCS_IC,
            SETUP_VALUES.TGT1_IR_SIZE,
            SETUP_VALUES.TGT2_IR_SIZE,
            SETUP_VALUES.TGT_TWO_IC,
            SETUP_VALUES.SOJ_ONE_IC,
            SETUP_VALUES.SOJ_TWO_IC,
            SETUP_VALUES.TGT_ECM_POWER_IC,
            SETUP_VALUES.TGT_ECM_TECH_IC,
            SETUP_VALUES.SOJ_ECM_TECH_IC);

        LOG_INTERVAL := SETUP_VALUES.LOG_INTERVAL;
        ENVIRONMENT.SET_TIME(0.0);
        INTEGRATION.SET_TIME_STEP_SIZE(SETUP_VALUES.FRAME_TIME);
        NEXT_LOGGING_FRAME := 1;
        FRAME_NUMBER := 0;
        INITIALIZE_SIMULATION(SETUP_VALUES.OUTPUT_FILE);
        INTEGRATION.INITIALIZE;

        while not END_CONDITION_MET loop
            FRAME_NUMBER := FRAME_NUMBER + 1;

            if FRAME_NUMBER = NEXT_LOGGING_FRAME then
                LOG_DATA;
                NEXT_LOGGING_FRAME := NEXT_LOGGING_FRAME + LOG_INTERVAL;
            end if;

            INTEGRATION.ADVANCE_TIME;

            if END_CONDITION_MET then
                LOG_DATA;
                SHOW_TERMINAL_CONDITIONS;
            end if;

            CHECK_PAUSE;

        end loop;

        END_OF_RUN;
    end SIMULATION_MAIN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function NUMBER_OF_STATE_VARIABLES
        return integer is
    begin
        return 14;
    end NUMBER_OF_STATE_VARIABLES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function END_CONDITION_MET
        return boolean is
    begin
        if TERMINATE_FLAG or MISSILE.END_CONDITIONS_MET then
            return true;
        else
            return false;
        end if;
    end END_CONDITION_MET;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure INITIALIZE_SYSTEM is
    begin
        SYSTEM_SPECIFIC.INIT_VIDEO;
        USER_INTERFACE.SHOW_TITLE_SCREEN;
    end INITIALIZE_SYSTEM;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure GET_DERIVATIVES (
        DERIVATIVES : out VECTOR) is
    begin
        DERIVATIVES(1..6)  := MISSILE.GET_DERIVATIVES;
        DERIVATIVES(7..9) := LAUNCHER.GET_DERIVATIVES;
        DERIVATIVES(10..12) := TARGETS.GET_DERIVATIVES;
    end GET_DERIVATIVES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure GET_STATES (
        STATES : out VECTOR) is
    begin
        STATES(1..6)  := MISSILE.GET_STATES;
        STATES(7..9) := LAUNCHER.GET_STATES;
        STATES(10..12) := TARGETS.GET_STATES;
    end GET_STATES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure PUT_STATES (
        STATES : in VECTOR) is
    begin
        MISSILE_STATES := STATES(1..6);
        MISSILE.PUT_STATES(MISSILE_STATES);
        LAUNCHER_STATES := STATES(7..9);
        TARGET_STATES := STATES(10..12);
        LAUNCHER.PUT_STATES(LAUNCHER_STATES);
        TARGETS.PUT_STATES(TARGET_STATES);
    end PUT_STATES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure COMPUTE_DERIVATIVES is
    begin
        LAUNCHER.COMPUTE;
        TARGETS.COMPUTE;
        MISSILE.COMPUTE;
    end COMPUTE_DERIVATIVES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure END_OF_RUN is
    begin
        if LOGGING_TO_DISK then
            Text_io.Close(LOG_DATA_FILE);
        end if;
    end END_OF_RUN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure GET_TIMES(TIME_ARRAY : out VECTOR; NUMBER_OF_VALUES : out
        integer) is
        NUMBER_VARIABLES : integer;
        DATA_RECORD : LOGGED_DATA_PTR_TYPE;
    begin
        DATA_RECORD := FIRST_DATA_RECORD;
        NUMBER_VARIABLES := 1;
        while (DATA_RECORD /= null) loop
            TIME_ARRAY(NUMBER_VARIABLES) := DATA_RECORD.Time;
            DATA_RECORD := DATA_RECORD.NEXT_RECORD;
            NUMBER_VARIABLES := NUMBER_VARIABLES + 1;
        end loop;
        NUMBER_OF_VALUES := NUMBER_VARIABLES - 1;
    end GET_TIMES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure GET_VARIABLE(WHICH_VARIABLE : in integer; RETURN_VARIABLE :
        out VECTOR; NUMBER_OF_VALUES : out integer) is
        NUMBER_VARIABLES : integer;
        DATA_RECORD : LOGGED_DATA_PTR_TYPE;
    begin
        DATA_RECORD := FIRST_DATA_RECORD;
        NUMBER_VARIABLES := 1;
        while (DATA_RECORD /= null) loop
            if (WHICH_VARIABLE >= 50) then
                RETURN_VARIABLE(NUMBER_VARIABLES) :=
                DATA_RECORD.DATA.LAUNCHER_DATA.REAL_VALUE(WHICH_VARIABLE-50);
                 -- !!! FIX LAUNCHER_DATA Also see PLOT.ADA
            else
                RETURN_VARIABLE(NUMBER_VARIABLES) :=
                DATA_RECORD.DATA.MISSILE_DATA.REAL_VALUE(WHICH_VARIABLE);
            end if;
            DATA_RECORD := DATA_RECORD.NEXT_RECORD;
            NUMBER_VARIABLES := NUMBER_VARIABLES + 1;
        end loop;
        NUMBER_OF_VALUES := NUMBER_VARIABLES - 1;
    end GET_VARIABLE;
end APPLICATION;
--------------------------------------------------------------------------------
-- Application Package Subunit
--
-- This package contains the application specific functions called by the
-- general Executive Package.
--
--------------------------------------------------------------------------------

separate (APPLICATION)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure LOG_DATA is
begin
    LOG_RECORD.MISSILE_DATA := MISSILE.LOG_DATA;
    LOG_RECORD.LAUNCHER_DATA := LAUNCHER.LOG_DATA;
    LOG_RECORD.TARGET_DATA := TARGETS.LOG_DATA;

    begin
        if not MEMORY_FULL then
            if FIRST_DATA_RECORD = null then
                FIRST_DATA_RECORD := new
                LOGGED_DATA'(time => environment.time,
                 DATA => LOG_RECORD, NEXT_RECORD => null);
                CURRENT_DATA_RECORD := FIRST_DATA_RECORD;
            else
                CURRENT_DATA_RECORD.NEXT_RECORD := new
                LOGGED_DATA'(time => environment.time,
                 DATA => LOG_RECORD, NEXT_RECORD => null);
                CURRENT_DATA_RECORD := CURRENT_DATA_RECORD.NEXT_RECORD;
            end if;
        end if;
    exception
        when STORAGE_ERROR =>
            MEMORY_FULL := true;
            SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
            SYSTEM_SPECIFIC.PUT_STRING(20, 41,
                "Data memory full at time :            ");
            SYSTEM_SPECIFIC.PUT_REAL(20,67, ENVIRONMENT.Time);
            SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
    end;

    if LOGGING_TO_DISK then
        REAL_IO.Put(LOG_DATA_FILE, ENVIRONMENT.Time, AFT => 3);
        Text_io.Put(LOG_DATA_FILE, ",");

        for I in 1..34 loop
            REAL_IO.Put(LOG_DATA_FILE, LOG_RECORD.MISSILE_DATA.REAL_VALUE(I),
                AFT => 3);
            Text_io.Put(LOG_DATA_FILE, ",");
        end loop;
    for I in 1..12 loop
        REAL_IO.Put(LOG_DATA_FILE, LOG_RECORD.LAUNCHER_DATA.REAL_VALUE(I),
            AFT => 3);
        Text_io.Put(LOG_DATA_FILE, ",");
    end loop;

    Text_io.Put(LOG_DATA_FILE, integer'image(GUIDANCE_PHASE_TYPE'pos(
                LOG_RECORD.MISSILE_DATA.GUIDANCE_PHASE)));
    Text_io.Put(LOG_DATA_FILE, ",");

    Text_io.Put(LOG_DATA_FILE, integer'image(RF_PHASE_TYPE1'pos(
                LOG_RECORD.MISSILE_DATA.RF_PHASE_1)));
    Text_io.Put(LOG_DATA_FILE, ",");

    Text_io.Put(LOG_DATA_FILE, integer'image(RF_PHASE_TYPE2'pos(
                LOG_RECORD.MISSILE_DATA.RF_PHASE_2)));
    Text_io.Put(LOG_DATA_FILE, ",");

    Text_io.Put(LOG_DATA_FILE, integer'image(IR_PHASE_TYPE'pos(
                LOG_RECORD.MISSILE_DATA.IR_PHASE)));
    Text_io.Put(LOG_DATA_FILE, ",");

    Text_io.Put(LOG_DATA_FILE, integer'image(boolean'pos(
                LOG_RECORD.MISSILE_DATA.RADOME_OFF)));
    Text_io.Put(LOG_DATA_FILE, ",");

    Text_io.Put(LOG_DATA_FILE, integer'image(PROPUL_TYPE'pos(
                LOG_RECORD.MISSILE_DATA.PROPULSION_PHASE)));

    Text_io.New_line(LOG_DATA_FILE);
end if;

SYSTEM_SPECIFIC.PUT_REAL( 1,29, REAL(Clock - START_TIME));
SYSTEM_SPECIFIC.PUT_REAL( 2,29, ENVIRONMENT.Time);
SYSTEM_SPECIFIC.PUT_REAL( 3,29,
    LOG_RECORD.MISSILE_DATA.REAL_VALUE(11)); -- Time to go
SYSTEM_SPECIFIC.PUT_STRING( 5,30,
    RF_MODE(LOG_RECORD.MISSILE_DATA.RF_PHASE_2)); -- RF seeker mode
SYSTEM_SPECIFIC.PUT_STRING( 6,30,
    IR_MODE(LOG_RECORD.MISSILE_DATA.IR_PHASE)); -- IR seeker mode
SYSTEM_SPECIFIC.PUT_REAL( 7,29, LOG_RECORD.MISSILE_DATA.REAL_VALUE(14));
-- Az Gimbal Angle
SYSTEM_SPECIFIC.PUT_REAL( 8,29, LOG_RECORD.MISSILE_DATA.REAL_VALUE(13));
-- El Gimbal Angle
SYSTEM_SPECIFIC.PUT_REAL( 9,29, LIMIT(LOG_RECORD.MISSILE_DATA.
        REAL_VALUE(16),9999.9999), 4); -- Az LOS Rate
SYSTEM_SPECIFIC.PUT_REAL(10,29, LIMIT(LOG_RECORD.MISSILE_DATA.
        REAL_VALUE(15),9999.9999), 4); -- El LOS Rate

SYSTEM_SPECIFIC.PUT_STRING(12,30, GUIDANCE_MODE(
        LOG_RECORD.MISSILE_DATA.GUIDANCE_PHASE)); -- Guidance mode
SYSTEM_SPECIFIC.PUT_REAL(13,29,
    LOG_RECORD.MISSILE_DATA.REAL_VALUE(18) / G, 4); -- Az Acc Cmd
SYSTEM_SPECIFIC.PUT_REAL(14,29,
    LOG_RECORD.MISSILE_DATA.REAL_VALUE(19) / G, 4); -- El Acc Cmd

SYSTEM_SPECIFIC.PUT_STRING(16,30, PROPULSION_MODE(
        LOG_RECORD.MISSILE_DATA.PROPULSION_PHASE)); -- Propulsion mode
SYSTEM_SPECIFIC.PUT_REAL(17,29,
    LOG_RECORD.MISSILE_DATA.REAL_VALUE(24), 1); -- Thrust

SYSTEM_SPECIFIC.PUT_REAL(19,29,
    LOG_RECORD.MISSILE_DATA.REAL_VALUE(23)); -- Msl Weight
SYSTEM_SPECIFIC.PUT_REAL(20,29, LOG_RECORD.MISSILE_DATA.REAL_VALUE(26));
-- Msl Alpha
SYSTEM_SPECIFIC.PUT_REAL(21,29, LOG_RECORD.MISSILE_DATA.REAL_VALUE(27));
-- Msl Beta

SYSTEM_SPECIFIC.PUT_REAL( 1,67,
        LOG_RECORD.MISSILE_DATA.REAL_VALUE(8), 3); -- Mach
    SYSTEM_SPECIFIC.PUT_REAL( 2,67,
        LOG_RECORD.MISSILE_DATA.REAL_VALUE(5), 1); -- Velocity
    SYSTEM_SPECIFIC.PUT_REAL( 3,67,
        LOG_RECORD.MISSILE_DATA.REAL_VALUE(3), 1); -- Altitude
    SYSTEM_SPECIFIC.PUT_REAL( 4,67,
        LOG_RECORD.MISSILE_DATA.REAL_VALUE(17)); -- Altitude Rate
    SYSTEM_SPECIFIC.PUT_REAL( 5,67, LOG_RECORD.MISSILE_DATA.REAL_VALUE(7));
    -- Pitch Angle
    SYSTEM_SPECIFIC.PUT_REAL( 6,67, LOG_RECORD.MISSILE_DATA.REAL_VALUE(6));
    -- Yaw Angle
    SYSTEM_SPECIFIC.PUT_REAL( 7,67, LOG_RECORD.
        MISSILE_DATA.REAL_VALUE(1) / FEET_PER_NMI , 1);-- Msl Downrange
    SYSTEM_SPECIFIC.PUT_REAL( 8,67, LOG_RECORD.
        MISSILE_DATA.REAL_VALUE(2) / FEET_PER_NMI , 1); -- Msl Crossrange
    SYSTEM_SPECIFIC.PUT_REAL(10,67,
        LOG_RECORD.MISSILE_DATA.REAL_VALUE(20) / G, 4); -- X Axis Accel
    SYSTEM_SPECIFIC.PUT_REAL(11,67,
        LOG_RECORD.MISSILE_DATA.REAL_VALUE(21) / G, 4); -- Y Axis Accel
    SYSTEM_SPECIFIC.PUT_REAL(12,67,
        LOG_RECORD.MISSILE_DATA.REAL_VALUE(22) / G, 4); -- Z Axis Accel

    SYSTEM_SPECIFIC.PUT_REAL(15,67, LOG_RECORD.
        MISSILE_DATA.REAL_VALUE(9) / FEET_PER_NMI , 1); -- Msl-Tgt Range

    SYSTEM_SPECIFIC.PUT_REAL(14,67, LOG_RECORD.
        MISSILE_DATA.REAL_VALUE(4) / FEET_PER_NMI , 1); -- Lnchr-Tgt Range
    SYSTEM_SPECIFIC.PUT_REAL(17,67,
        LOG_RECORD.LAUNCHER_DATA.REAL_VALUE(11), 3); -- Tgt Mach
    SYSTEM_SPECIFIC.PUT_REAL(16,67,
        LOG_RECORD.MISSILE_DATA.REAL_VALUE(10), 1); -- Tgt Range Rate
    SYSTEM_SPECIFIC.PUT_REAL(18,67,
        LOG_RECORD.LAUNCHER_DATA.REAL_VALUE(6), 1); -- Tgt Altitude
    SYSTEM_SPECIFIC.PUT_REAL(19,67, LOG_RECORD.LAUNCHER_DATA.
        REAL_VALUE(12), 1); -- Tgt heading ang
end LOG_DATA;
--------------------------------------------------------------------------------
-- User Interface Package Specification
--
-- This package contains all the procedures that perform input/output between
-- the program and the user via the screen and keyboard.
--
--------------------------------------------------------------------------------

with MATH; use MATH;

package USER_INTERFACE is

    procedure MAIN;

    procedure SHOW_TITLE_SCREEN;

    procedure DRAW_RUNTIME_BORDER;

    procedure SETUP_RUNTIME_SCREEN;

    task KEYBOARD_HANDLER is
        entry KEY_AVAILABLE(KEY_IN_BUFFER : out boolean);

        entry GET_KEY_WAIT(KEY : out character);

        entry GET_KEY_NOWAIT(KEY : out character; KEY_VALID : out boolean);
    end KEYBOARD_HANDLER;

end USER_INTERFACE;
--------------------------------------------------------------------------------
-- User Interface Package Body
--
-- This package contains all the procedures that perform input/output between
-- the program and the user via the screen and keyboard.
--
-------------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with SYSTEM_SPECIFIC;
with APPLICATION;
with MISSILE;
with PLOT;
with Text_io;
with REAL_IO;

package body USER_INTERFACE is
    MESSAGE_DISPLAYED : boolean := false; -- Message on at screen bottom
    MENU_START_ROW : constant := 6;
    DATA_COLUMN : constant := 42;
    ITEM_TEXT_SIZE : constant := 30;
    MENU_TEXT_SIZE : constant := 30;
    STRING_SIZE : constant := 60;

    type STRING_ARRAY is array(positive range <>) of string(1..STRING_SIZE);

    TEMP_STRING : string(1..ITEM_TEXT_SIZE);

    QUIT_STRING : string(1..MENU_TEXT_SIZE) := "Quit program                  ";
    RETURN_STRING : string(1..MENU_TEXT_SIZE):="Return to previous menu       ";
    BLANK_STRING : string(1..STRING_SIZE) :=
    "                                                            ";

    ---------------------------------------------------------------------------

    type MENU_ITEM_KIND is (SUBMENU, DATA_ITEM, ACTION);
    type DATA_ITEM_KIND is (NONE, TEXT, FLOATING_PT, YES_NO, INT_GUIDANCE,
        TGT_IR_SIZE, MANEUVER, MANEUVER_START, AIRCRAFT_KIND, LAUNCHER,
        SSJ_ECM, SOJ_ECM);
    type ACTION_TYPE is (LOAD_DATA, SAVE_DATA, START_RUN, GRP1_GRAPH1,
         GRP1_GRAPH2, GRP1_GRAPH3, GRP1_GRAPH4, GRP1_GRAPH5, GRP1_GRAPH6,
         GRP1_GRAPH7, GRP2_GRAPH1, GRP2_GRAPH2, GRP2_GRAPH3, GRP2_GRAPH4,
         GRP2_GRAPH5, GRP3_GRAPH1, GRP3_GRAPH2, GRP3_GRAPH3, GRP4_GRAPH1,
         GRP4_GRAPH2, GRP4_GRAPH3, GRP4_GRAPH4, GRP4_GRAPH5, GRP4_GRAPH6,
         GRP4_GRAPH7);

    type MENU_TYPE;
    type MENU_POINTER is access MENU_TYPE;

    type MENU_ITEM_TYPE(ITEM_KIND : MENU_ITEM_KIND;
        DATA_TYPE : DATA_ITEM_KIND);
    type MENU_ITEM_POINTER is access MENU_ITEM_TYPE;

    type MENU_TYPE is
    record
        TITLE : string(1..STRING_SIZE);
        FIRST_ITEM : MENU_ITEM_POINTER;
        ROWS_BETWEEN_ITEMS : positive; -- 1 = no gap, 2 = 1 blank line, etc.
    end record;

    type MENU_ITEM_TYPE(ITEM_KIND : MENU_ITEM_KIND; DATA_TYPE :
        DATA_ITEM_KIND) is
    record
        TITLE : string(1..MENU_TEXT_SIZE);
        NEXT_ITEM : MENU_ITEM_POINTER;

        case ITEM_KIND is
            when SUBMENU =>
                NEXT_MENU : MENU_POINTER;
            when DATA_ITEM =>
                PROMPT : string(1..MENU_TEXT_SIZE);
                case DATA_TYPE is
                    when NONE =>
                        null;
                    when TEXT =>
                        TEXT_VALUE : string(1..MENU_TEXT_SIZE);
                    when FLOATING_PT =>
                        REAL_VALUE : REAL;
                        MAX_VALUE : REAL;
                        MIN_VALUE : REAL;
                    when YES_NO =>
                        YES_NO_VALUE : YES_NO_TYPE;
                    when INT_GUIDANCE =>
                        INT_GUIDANCE_VALUE : INT_GUIDANCE_TYPE;
                    when TGT_IR_SIZE =>
                        TGT_IR_SIZE_VALUE : TGT_IR_SIZE_TYPE;
                    when MANEUVER =>
                        MANEUVER_VALUE : MANEUVER_TYPE;
                    when MANEUVER_START =>
                        MANEUVER_START_VALUE : MANEUVER_START_TYPE;
                    when AIRCRAFT_KIND =>
                        AIRCRAFT_KIND_VALUE : AIRCRAFT_TYPE;
                    when LAUNCHER =>
                        LAUNCHER_VALUE : LAUNCHER_TYPE;
                    when SSJ_ECM =>
                        SSJ_ECM_VALUE : SSJ_ECM_TYPE;
                    when SOJ_ECM =>
                        SOJ_ECM_VALUE : SOJ_ECM_TYPE;
                end case;
            when ACTION =>
                ACTION_KIND : ACTION_TYPE;
        end case;
    end record;

    ----------------------------------------------------------------------------

    MAIN_MENU, FILE_MENU, LAUNCHER_MENU, TGT_MENU, GRAPH_MENU : MENU_POINTER;
    GRAPH_GRP1_MENU, GRAPH_GRP2_MENU, GRAPH_GRP3_MENU, GRAPH_GRP4_MENU,
    TGT1_MENU, TGT2_MENU, SOJ1_MENU, SOJ2_MENU : MENU_POINTER;

    type MENU_ITEM_ARRAY is array (positive range <>) of MENU_ITEM_POINTER;

    MAIN_MENU_ITEMS : MENU_ITEM_ARRAY(1..5);
    FILE_MENU_ITEMS : MENU_ITEM_ARRAY(1..7);
    LAUNCHER_MENU_ITEMS : MENU_ITEM_ARRAY(1..6);
    TGT_MENU_ITEMS : MENU_ITEM_ARRAY(1..7);
    TGT1_MENU_ITEMS : MENU_ITEM_ARRAY(1..15);
    TGT2_MENU_ITEMS : MENU_ITEM_ARRAY(1..7);
    SOJ1_MENU_ITEMS : MENU_ITEM_ARRAY(1..13);
    SOJ2_MENU_ITEMS : MENU_ITEM_ARRAY(1..13);
    GRAPH_MENU_ITEMS : MENU_ITEM_ARRAY(1..4);
    GRAPH_GRP1_MENU_ITEMS :  MENU_ITEM_ARRAY(1..8);
    GRAPH_GRP2_MENU_ITEMS :  MENU_ITEM_ARRAY(1..5);
    GRAPH_GRP3_MENU_ITEMS :  MENU_ITEM_ARRAY(1..3);
    GRAPH_GRP4_MENU_ITEMS :  MENU_ITEM_ARRAY(1..7);

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SHOW_TITLE_SCREEN is separate;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure GET_TEXT(PROMPT_STRING : in string; TEXT_STRING : in out string)is
        TEMP_STRING : string(TEXT_STRING'range);
        INPUT_VALID : boolean := false;
    begin
        for I in TEMP_STRING'range loop
            TEMP_STRING(I) := ' ';
        end loop;
        SYSTEM_SPECIFIC.PUT_STRING(22, 10, PROMPT_STRING);
        SYSTEM_SPECIFIC.MOVE_CURSOR(22,DATA_COLUMN);
        SYSTEM_SPECIFIC.TURN_CURSOR_ON;
        SYSTEM_SPECIFIC.INPUT_STRING(TEMP_STRING);
        SYSTEM_SPECIFIC.TURN_CURSOR_OFF;
        SYSTEM_SPECIFIC.PUT_STRING(22, 10, BLANK_STRING);
        for I in TEMP_STRING'range loop
            if TEMP_STRING(I) /= ' ' then
                INPUT_VALID := true;
            end if;
        end loop;
        if INPUT_VALID then
            TEXT_STRING := TEMP_STRING;
        end if;
    end GET_TEXT;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure GET_REAL(PROMPT_STRING : in string; NUMBER : in out REAL) is
        TEMP_NUMBER : REAL;
        EXPONENT : REAL;
        CURRENT  : positive;
        SIGN  : integer;
    begin
        REAL_IO.Put(TEMP_STRING,NUMBER,AFT=>10,EXP=>0);
        GET_TEXT(PROMPT_STRING, TEMP_STRING);

        TEMP_NUMBER := 0.0;
        CURRENT := TEMP_STRING'first;
        SIGN := 1;

        for I in TEMP_STRING'range loop -- Skip tabs and spaces
            if TEMP_STRING(I) /= ' ' and TEMP_STRING(I) /= Ascii.HT then
                exit;
            end if;
            CURRENT := CURRENT + 1;
        end loop;

        if TEMP_STRING(CURRENT) = '+' then
            CURRENT := CURRENT + 1;
        end if;

        if TEMP_STRING(CURRENT) = '-' then
            SIGN := -1;
            CURRENT := CURRENT + 1;
        end if;

        while TEMP_STRING(CURRENT) >= '0' and TEMP_STRING(CURRENT) <= '9' loop
            TEMP_NUMBER := TEMP_NUMBER * 10.0;
            TEMP_NUMBER := TEMP_NUMBER + REAL(
                character'pos(TEMP_STRING(CURRENT)) - character'pos('0'));
            CURRENT := CURRENT + 1;
        end loop;

        if TEMP_STRING(CURRENT) = '.' then
            CURRENT := CURRENT + 1;
            EXPONENT := 0.1;

            while TEMP_STRING(CURRENT) >= '0' and TEMP_STRING(CURRENT) <= '9'
            loop
                TEMP_NUMBER := TEMP_NUMBER + REAL(
                    character'pos(TEMP_STRING(CURRENT)) -
                    character'pos('0')) * EXPONENT;
                EXPONENT := EXPONENT / 10.0;
                CURRENT := CURRENT + 1;
            end loop;
        end if;

        NUMBER := TEMP_NUMBER * REAL(SIGN);

    exception
        when others =>
        return;
    end GET_REAL;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function SAVE_DATA_FILE
        return boolean is

        DATA_FILE : Text_io.File_type;

        procedure OUTPUT_MENU(MENU : in MENU_POINTER) is
            ITEM : MENU_ITEM_POINTER := MENU.FIRST_ITEM;
        begin
            while ITEM /= null loop
                if ITEM.ITEM_KIND = DATA_ITEM then
                    case ITEM.DATA_TYPE is
                        when NONE =>
                            null;
                        when TEXT =>
                            Text_io.Put_line(DATA_FILE, ITEM.TEXT_VALUE);
                        when FLOATING_PT =>
                            REAL_IO.Put(DATA_FILE, ITEM.REAL_VALUE);
                            Text_io.New_line(DATA_FILE);
                        when YES_NO =>
                            Text_io.Put_line(DATA_FILE, YES_NO_TYPE'image(
                                    ITEM.YES_NO_VALUE));
                        when INT_GUIDANCE =>
                            Text_io.Put_line(DATA_FILE, INT_GUIDANCE_TYPE'image(
                                    ITEM.INT_GUIDANCE_VALUE));
                        when TGT_IR_SIZE =>
                            Text_io.Put_line(DATA_FILE, TGT_IR_SIZE_TYPE'image(
                                    ITEM.TGT_IR_SIZE_VALUE));
                        when MANEUVER =>
                            Text_io.Put_line(DATA_FILE, MANEUVER_TYPE'image(
                                    ITEM.MANEUVER_VALUE));
                        when MANEUVER_START =>
                            Text_io.Put_line(DATA_FILE,
                                MANEUVER_START_TYPE'image(
                                    ITEM.MANEUVER_START_VALUE));
                        when AIRCRAFT_KIND =>
                            Text_io.Put_line(DATA_FILE,
                                AIRCRAFT_TYPE'image(
                                    ITEM.AIRCRAFT_KIND_VALUE));
                        when LAUNCHER =>
                            Text_io.Put_line(DATA_FILE,
                                LAUNCHER_TYPE'image(
                                    ITEM.LAUNCHER_VALUE));
                        when SSJ_ECM =>
                            Text_io.Put_line(DATA_FILE,
                                SSJ_ECM_TYPE'image(ITEM.SSJ_ECM_VALUE));
                        when SOJ_ECM =>
                            Text_io.Put_line(DATA_FILE,
                                SOJ_ECM_TYPE'image(ITEM.SOJ_ECM_VALUE));
                    end case;
                elsif ITEM.ITEM_KIND = SUBMENU then
                    OUTPUT_MENU(ITEM.NEXT_MENU);
                end if;

                ITEM := ITEM.NEXT_ITEM;
            end loop;
        end OUTPUT_MENU;

    begin
        Text_io.Create(DATA_FILE, Text_io.Out_file,
            FILE_MENU_ITEMS(1).TEXT_VALUE);
        Text_io.Put_line(DATA_FILE, MISSILE.ID_STRING);

        OUTPUT_MENU(MAIN_MENU);
        Text_io.Close(DATA_FILE);
        return true;
    exception
        when others =>
            SYSTEM_SPECIFIC.PUT_STRING(22, 10, "ERROR during file save");
            MESSAGE_DISPLAYED := true;
            if Text_io.Is_open(DATA_FILE) then
                Text_io.Close(DATA_FILE);
            end if;
            return false;
    end SAVE_DATA_FILE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function LOAD_DATA_FILE
        return boolean is

        DATA_FILE  : Text_io.File_type;
        ID_MISMATCH : exception;

        procedure INPUT_MENU(MENU : in MENU_POINTER) is
            ITEM : MENU_ITEM_POINTER := MENU.FIRST_ITEM;
            TEMP_LINE : string(1..MENU_TEXT_SIZE);
            LAST : natural;
        begin
            while ITEM /= null loop
                if ITEM.ITEM_KIND = DATA_ITEM then
                    case ITEM.DATA_TYPE is
                        when NONE =>
                            null;
                        when TEXT =>
                            Text_io.Get_line(DATA_FILE, ITEM.TEXT_VALUE, LAST);
                            Text_io.Skip_line(DATA_FILE);
                        when FLOATING_PT =>
                            REAL_IO.Get(DATA_FILE, ITEM.REAL_VALUE);
                            Text_io.Skip_line(DATA_FILE);
                        when YES_NO =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.YES_NO_VALUE :=
                            YES_NO_TYPE'value(TEMP_LINE(1..LAST));
                        when INT_GUIDANCE =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.INT_GUIDANCE_VALUE :=
                            INT_GUIDANCE_TYPE'value(TEMP_LINE(1..LAST));
                        when TGT_IR_SIZE =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.TGT_IR_SIZE_VALUE :=
                            TGT_IR_SIZE_TYPE'value(TEMP_LINE(1..LAST));
                        when MANEUVER =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.MANEUVER_VALUE :=
                            MANEUVER_TYPE'value(TEMP_LINE(1..LAST));
                        when MANEUVER_START =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.MANEUVER_START_VALUE :=
                            MANEUVER_START_TYPE'value(TEMP_LINE(1..LAST));
                        when AIRCRAFT_KIND =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.AIRCRAFT_KIND_VALUE :=
                            AIRCRAFT_TYPE'value(TEMP_LINE(1..LAST));
                        when LAUNCHER =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.LAUNCHER_VALUE :=
                            LAUNCHER_TYPE'value(TEMP_LINE(1..LAST));
                        when SSJ_ECM =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.SSJ_ECM_VALUE :=
                            SSJ_ECM_TYPE'value(TEMP_LINE(1..LAST));
                        when SOJ_ECM =>
                            Text_io.Get_line(DATA_FILE, TEMP_LINE, LAST);
                            ITEM.SOJ_ECM_VALUE :=
                            SOJ_ECM_TYPE'value(TEMP_LINE(1..LAST));
                end case;
            elsif ITEM.ITEM_KIND = SUBMENU then
                INPUT_MENU(ITEM.NEXT_MENU);
            end if;

            ITEM := ITEM.NEXT_ITEM;
        end loop;
    end INPUT_MENU;

begin
    Text_io.Open(DATA_FILE, Text_io.In_file, FILE_MENU_ITEMS(1).TEXT_VALUE);

    Text_io.Get(DATA_FILE, TEMP_STRING);
    Text_io.Skip_line(DATA_FILE);

    for I in MISSILE.ID_STRING'range loop
        if MISSILE.ID_STRING(I) /= TEMP_STRING(I) then
            raise ID_MISMATCH;
        end if;
    end loop;

    INPUT_MENU(MAIN_MENU);

    Text_io.Close(DATA_FILE);
    return true;

exception
    when others =>
        SYSTEM_SPECIFIC.PUT_STRING(22, 10, "ERROR during file load");
        MESSAGE_DISPLAYED := true;
        if Text_io.Is_open(DATA_FILE) then
            Text_io.Close(DATA_FILE);
        end if;
        return false;
end LOAD_DATA_FILE;
------------------------------------------------------------------------------
------------------------------------------------------------------------------
procedure START_RUN is
    SETUP_VALUES : SETUP_VALUES_TYPE;
    TEMP_REAL : REAL; -- Needed for Meridian V4.0 bug
begin
    --        File operations parameters

    SETUP_VALUES.OUTPUT_FILE := FILE_MENU_ITEMS(4).TEXT_VALUE;
    SETUP_VALUES.LOG_DATA := FILE_MENU_ITEMS(5).YES_NO_VALUE;
    SETUP_VALUES.FRAME_TIME := FILE_MENU_ITEMS(6).REAL_VALUE;
    SETUP_VALUES.LOG_INTERVAL := INTEGER(FILE_MENU_ITEMS(7).REAL_VALUE);

    --        Launch aircraft parameters:
    SETUP_VALUES.INT_TYPE_IC:=LAUNCHER_MENU_ITEMS(1).AIRCRAFT_KIND_VALUE;
    SETUP_VALUES.LAUNCH_TYPE := LAUNCHER_MENU_ITEMS(2).LAUNCHER_VALUE;
    TEMP_REAL := LAUNCHER_MENU_ITEMS(3).REAL_VALUE;
    SETUP_VALUES.INT_ALTITUDE_IC := TEMP_REAL;
    SETUP_VALUES.INT_MACH_IC := LAUNCHER_MENU_ITEMS(4).REAL_VALUE;
    SETUP_VALUES.INT_LEAD_ANGLE_IC := LAUNCHER_MENU_ITEMS(5).REAL_VALUE;
    SETUP_VALUES.INT_GUIDANCE := LAUNCHER_MENU_ITEMS(6).INT_GUIDANCE_VALUE;

    --        Target parameters:

    SETUP_VALUES.TGT_TWO_IC:=TGT_MENU_ITEMS(2).YES_NO_VALUE;
    SETUP_VALUES.SOJ_ONE_IC:=TGT_MENU_ITEMS(4).YES_NO_VALUE;
    SETUP_VALUES.SOJ_TWO_IC:=TGT_MENU_ITEMS(6).YES_NO_VALUE;
    TEMP_REAL := TGT1_MENU_ITEMS(1).REAL_VALUE;
    SETUP_VALUES.TGT_ALTITUDE_IC(1) := 1000.0*TEMP_REAL;
    TEMP_REAL := TGT2_MENU_ITEMS(1).REAL_VALUE;
    SETUP_VALUES.TGT_ALTITUDE_IC(2) := 1000.0*TEMP_REAL;
    TEMP_REAL := SOJ1_MENU_ITEMS(1).REAL_VALUE;
    SETUP_VALUES.TGT_ALTITUDE_IC(3) := 1000.0*TEMP_REAL;
    TEMP_REAL := SOJ2_MENU_ITEMS(1).REAL_VALUE;
```

```
        SETUP_VALUES.TGT_ALTITUDE_IC(4) := 1000.0*TEMP_REAL;
        SETUP_VALUES.SOJ_ANGLE_IC(1):=SOJ1_MENU_ITEMS(3).REAL_VALUE;
        SETUP_VALUES.SOJ_ANGLE_IC(2):=SOJ2_MENU_ITEMS(3).REAL_VALUE;
        SETUP_VALUES.TGT_MACH_IC := TGT1_MENU_ITEMS(2).REAL_VALUE;
        SETUP_VALUES.TGT_ASPECT_IC := TGT1_MENU_ITEMS(3).REAL_VALUE;
        SETUP_VALUES.TGT_RANGE_IC(1) := FEET_PER_NMI*
        TGT1_MENU_ITEMS(4).REAL_VALUE;
        SETUP_VALUES.TGT_RANGE_IC(2) := TGT2_MENU_ITEMS(2).REAL_VALUE;
        SETUP_VALUES.TGT_RANGE_IC(3) := FEET_PER_NMI*
        SOJ1_MENU_ITEMS(2).REAL_VALUE;
        SETUP_VALUES.TGT_RANGE_IC(4) := FEET_PER_NMI*
        SOJ2_MENU_ITEMS(2).REAL_VALUE;
        SETUP_VALUES.TGT2_ANGLE_IC:=TGT2_MENU_ITEMS(3).REAL_VALUE;
        TEMP_REAL:=TGT1_MENU_ITEMS(11).REAL_VALUE;
        SETUP_VALUES.TGT_BUILDUP_TIME_IC :=TEMP_REAL;
        SETUP_VALUES.TGT_RCS_IC(1) := 10.76*TGT1_MENU_ITEMS(5).REAL_VALUE;
        SETUP_VALUES.TGT_RCS_IC(2) := 10.76*TGT2_MENU_ITEMS(4).REAL_VALUE;
        SETUP_VALUES.TGT_ECM_TECH_IC(1):=TGT1_MENU_ITEMS(14).SSJ_ECM_VALUE;
        SETUP_VALUES.TGT_ECM_TECH_IC(2):=TGT2_MENU_ITEMS(6).SSJ_ECM_VALUE;
        SETUP_VALUES.SOJ_ECM_TECH_IC(1):=SOJ1_MENU_ITEMS(4).SOJ_ECM_VALUE;
        SETUP_VALUES.SOJ_ECM_TECH_IC(2):=SOJ2_MENU_ITEMS(4).SOJ_ECM_VALUE;
        SETUP_VALUES.TGT_ECM_POWER_IC(1):=TGT1_MENU_ITEMS(15).REAL_VALUE;
        SETUP_VALUES.TGT_ECM_POWER_IC(2):=TGT2_MENU_ITEMS(7).REAL_VALUE;
        SETUP_VALUES.TGT_ECM_POWER_IC(3):=SOJ1_MENU_ITEMS(5).REAL_VALUE;
        SETUP_VALUES.TGT_ECM_POWER_IC(4):=SOJ2_MENU_ITEMS(5).REAL_VALUE;
        SETUP_VALUES.TGT1_IR_SIZE := TGT1_MENU_ITEMS(6).TGT_IR_SIZE_VALUE;
        SETUP_VALUES.TGT2_IR_SIZE := TGT2_MENU_ITEMS(5).TGT_IR_SIZE_VALUE;
        SETUP_VALUES.MANEUVER_KIND := TGT1_MENU_ITEMS(7).MANEUVER_VALUE;
        SETUP_VALUES.TGT_TURN_G_IC := TGT1_MENU_ITEMS(8).REAL_VALUE;
        SETUP_VALUES.TURN_ON_PARAMETER :=
        TGT1_MENU_ITEMS(9).MANEUVER_START_VALUE;
        SETUP_VALUES.TGT_TURNON_VALUE_IC := TGT1_MENU_ITEMS(10).REAL_VALUE;
        SETUP_VALUES.TGT_TURN_ANG_IC := TGT1_MENU_ITEMS(12).REAL_VALUE;
        SETUP_VALUES.TGT_WEAVE_PER_IC := TGT1_MENU_ITEMS(13).REAL_VALUE;
        APPLICATION.SIMULATION_MAIN(SETUP_VALUES);
    end START_RUN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function DISPLAY_MENU(MENU : MENU_POINTER; ITEM_NUM : integer)
        return integer is

        ITEM : MENU_ITEM_POINTER;
        ROW : integer;
        NUM_ITEMS : integer;
    begin
        SYSTEM_SPECIFIC.CLEAR_SCREEN;
        SYSTEM_SPECIFIC.DRAW_BOX;

        SYSTEM_SPECIFIC.PUT_STRING( 2, 10, MENU.TITLE);

        if MENU = MAIN_MENU then
            SYSTEM_SPECIFIC.PUT_STRING(22, 5, "Use the up and down " &
                "arrow keys to move the cursor and then press ENTER");
        end if;

        ITEM := MENU.FIRST_ITEM;
        ROW := MENU_START_ROW;
        NUM_ITEMS := 0;
        while ITEM /= null loop
            if NUM_ITEMS = ITEM_NUM then
                SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
            end if;

            SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);

            if NUM_ITEMS = ITEM_NUM then
                SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
            end if;
            if ITEM.ITEM_KIND = DATA_ITEM then
                case ITEM.DATA_TYPE is
                    when NONE =>
                        null;
                    when TEXT =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            ITEM.TEXT_VALUE);
                    when FLOATING_PT =>
                        SYSTEM_SPECIFIC.PUT_REAL( ROW, DATA_COLUMN,
                            ITEM.REAL_VALUE);
                    when YES_NO =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            YES_NO_TYPE'image(ITEM.YES_NO_VALUE));
                    when INT_GUIDANCE =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            INT_GUIDANCE_TYPE'image(ITEM.INT_GUIDANCE_VALUE));
                    when TGT_IR_SIZE =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            TGT_IR_SIZE_TYPE'image(ITEM.TGT_IR_SIZE_VALUE));
                    when MANEUVER =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            MANEUVER_TYPE'image(ITEM.MANEUVER_VALUE));
                    when MANEUVER_START =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            MANEUVER_START_TYPE'image(
                                ITEM.MANEUVER_START_VALUE));
                    when AIRCRAFT_KIND =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            AIRCRAFT_TYPE'image(
                                ITEM.AIRCRAFT_KIND_VALUE));
                    when LAUNCHER =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            LAUNCHER_TYPE'image(
                                ITEM.LAUNCHER_VALUE));
                    when SSJ_ECM =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            SSJ_ECM_TYPE'image(ITEM.SSJ_ECM_VALUE));
                    when SOJ_ECM =>
                        SYSTEM_SPECIFIC.PUT_STRING( ROW, DATA_COLUMN,
                            SOJ_ECM_TYPE'image(ITEM.SOJ_ECM_VALUE));
                end case;
            end if;

            ROW := ROW + MENU.ROWS_BETWEEN_ITEMS;
            ITEM := ITEM.NEXT_ITEM;
            NUM_ITEMS := NUM_ITEMS + 1;
        end loop;

        if MENU = MAIN_MENU then
            SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, QUIT_STRING);
        else
            SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, RETURN_STRING);
        end if;

        return NUM_ITEMS;
    end DISPLAY_MENU;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure MANAGE_MENU(MENU : MENU_POINTER) is
        ITEM : MENU_ITEM_POINTER;
        ROW : integer;
        ITEM_NUM : integer := 0;
        NUM_ITEMS : integer;
        COMMAND : SYSTEM_SPECIFIC.MENU_COMMAND;
        STATUS_OK : boolean;
    begin
        NUM_ITEMS := DISPLAY_MENU(MENU, ITEM_NUM);

        ROW := MENU_START_ROW;
```

```
ITEM := MENU.FIRST_ITEM;
loop
    COMMAND := SYSTEM_SPECIFIC.GET_MENU_COMMAND;

    if MESSAGE_DISPLAYED then
        SYSTEM_SPECIFIC.PUT_STRING(22, 10, BLANK_STRING);
        MESSAGE_DISPLAYED := false;
    end if;

    case COMMAND is
        when SYSTEM_SPECIFIC.UP_ARROW =>
            if ITEM_NUM = NUM_ITEMS then
                if MENU = MAIN_MENU then
                    SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, QUIT_STRING);
                else
                    SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, RETURN_STRING);
                end if;
                ROW := ROW - MENU.ROWS_BETWEEN_ITEMS;
                ITEM_NUM := ITEM_NUM - 1;
                ITEM := MENU.FIRST_ITEM;
                for I in 0..ITEM_NUM-1 loop
                    ITEM := ITEM.NEXT_ITEM;
                end loop;
                SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
                SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);
                SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
            elsif ITEM_NUM = 0 then
                SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);
                ROW := MENU_START_ROW + NUM_ITEMS *
                MENU.ROWS_BETWEEN_ITEMS;
                ITEM_NUM := NUM_ITEMS;
                SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
                if MENU = MAIN_MENU then
                    SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, QUIT_STRING);
                else
                    SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, RETURN_STRING);
                end if;
                SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
            else
                SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);
                ROW := ROW - MENU.ROWS_BETWEEN_ITEMS;
                ITEM_NUM := ITEM_NUM - 1;
                ITEM := MENU.FIRST_ITEM;
                for I in 0..ITEM_NUM-1 loop
                    ITEM := ITEM.NEXT_ITEM;
                end loop;
                SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
                SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);
                SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
            end if;

        when SYSTEM_SPECIFIC.DOWN_ARROW =>
            if ITEM_NUM < NUM_ITEMS-1 then
                SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);
                ROW := ROW + MENU.ROWS_BETWEEN_ITEMS;
                ITEM_NUM := ITEM_NUM + 1;
                ITEM := ITEM.NEXT_ITEM;
                SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
                SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);
                SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
            elsif ITEM_NUM = NUM_ITEMS-1 then
                SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);
                ROW := ROW + MENU.ROWS_BETWEEN_ITEMS;
                ITEM_NUM := ITEM_NUM + 1;
                SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
                if MENU = MAIN_MENU then
                    SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, QUIT_STRING);
                else
                    SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, RETURN_STRING);
                end if;
                SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
            else -- ITEM_NUM = NUM_ITEMS
                if MENU = MAIN_MENU then
                    SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, QUIT_STRING);
                else
                    SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, RETURN_STRING);
                end if;
                ROW := MENU_START_ROW;
                ITEM := MENU.FIRST_ITEM;
                ITEM_NUM := 0;

                SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
                SYSTEM_SPECIFIC.PUT_STRING( ROW, 10, ITEM.TITLE);
                SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
            end if;
        when SYSTEM_SPECIFIC.ENTER_KEY =>
            if ITEM_NUM = NUM_ITEMS then
                if MENU = MAIN_MENU then
                    SYSTEM_SPECIFIC.QUIT_PROGRAM;
                else
                    return;
                end if;
            else
                case ITEM.ITEM_KIND is
                    when SUBMENU =>
                        MANAGE_MENU(ITEM.NEXT_MENU);
                        NUM_ITEMS := DISPLAY_MENU(MENU, ITEM_NUM);
                    when DATA_ITEM =>
                        case ITEM.DATA_TYPE is
                            when NONE =>
                                null;
                            when TEXT =>
                                GET_TEXT(ITEM.PROMPT, ITEM.TEXT_VALUE);
                                SYSTEM_SPECIFIC.PUT_STRING(ROW,
                                    DATA_COLUMN, ITEM.TEXT_VALUE);
                            when FLOATING_PT =>
                                GET_REAL(ITEM.PROMPT, ITEM.REAL_VALUE);
                                SYSTEM_SPECIFIC.PUT_REAL( ROW,
                                    DATA_COLUMN, ITEM.REAL_VALUE);
                            when YES_NO =>
                                if ITEM.YES_NO_VALUE /= YES_NO_TYPE'last
                                then
                                    ITEM.YES_NO_VALUE :=
                                    YES_NO_TYPE'succ(
                                        ITEM.YES_NO_VALUE);
                                else
                                    ITEM.YES_NO_VALUE :=
                                    YES_NO_TYPE'first;
                                end if;
                                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                                    DATA_COLUMN, BLANK_STRING(
                                        1..YES_NO_TYPE'width));
                                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                                    DATA_COLUMN, YES_NO_TYPE'image(
                                        ITEM.YES_NO_VALUE));
                            when INT_GUIDANCE =>
                                if ITEM.INT_GUIDANCE_VALUE /=
                                INT_GUIDANCE_TYPE'last then
                                    ITEM.INT_GUIDANCE_VALUE :=
                                    INT_GUIDANCE_TYPE'succ(
                                        ITEM.INT_GUIDANCE_VALUE);
                                else
                                    ITEM.INT_GUIDANCE_VALUE :=
                                    INT_GUIDANCE_TYPE'first;
                                end if;
                                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                                    DATA_COLUMN, BLANK_STRING(
                                        1..INT_GUIDANCE_TYPE'width));
```

```
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN,
        INT_GUIDANCE_TYPE'image(
            ITEM.INT_GUIDANCE_VALUE));
when TGT_IR_SIZE =>
    if ITEM.TGT_IR_SIZE_VALUE /=
    TGT_IR_SIZE_TYPE'last then
        ITEM.TGT_IR_SIZE_VALUE :=
        TGT_IR_SIZE_TYPE'succ(
            ITEM.TGT_IR_SIZE_VALUE);
    else
        ITEM.TGT_IR_SIZE_VALUE :=
        TGT_IR_SIZE_TYPE'first;
    end if;
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN, BLANK_STRING(
            1..TGT_IR_SIZE_TYPE'width));
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN,
        TGT_IR_SIZE_TYPE'image(
            ITEM.TGT_IR_SIZE_VALUE));
when MANEUVER =>
    if ITEM.MANEUVER_VALUE /=
    MANEUVER_TYPE'last then
        ITEM.MANEUVER_VALUE :=
        MANEUVER_TYPE'succ(
            ITEM.MANEUVER_VALUE);
    else
        ITEM.MANEUVER_VALUE :=
        MANEUVER_TYPE'first;
    end if;
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN, BLANK_STRING(
            1..MANEUVER_TYPE'width));
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN, MANEUVER_TYPE'image(
            ITEM.MANEUVER_VALUE));
when MANEUVER_START =>
    if ITEM.MANEUVER_START_VALUE /=
    MANEUVER_START_TYPE'last then
        ITEM.MANEUVER_START_VALUE :=
        MANEUVER_START_TYPE'succ(
            ITEM.MANEUVER_START_VALUE);
    else
        ITEM.MANEUVER_START_VALUE :=
        MANEUVER_START_TYPE'first;
    end if;
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN, BLANK_STRING(
            1..MANEUVER_START_TYPE'width));
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN,
        MANEUVER_START_TYPE'image(
            ITEM.MANEUVER_START_VALUE));
when AIRCRAFT_KIND =>
    if ITEM.AIRCRAFT_KIND_VALUE /=
    AIRCRAFT_TYPE'last then
        ITEM.AIRCRAFT_KIND_VALUE :=
        AIRCRAFT_TYPE'succ(
            ITEM.AIRCRAFT_KIND_VALUE);
    else
        ITEM.AIRCRAFT_KIND_VALUE :=
        AIRCRAFT_TYPE'first;
    end if;
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN, BLANK_STRING(
            1..AIRCRAFT_TYPE'width));
    SYSTEM_SPECIFIC.PUT_STRING( ROW,
        DATA_COLUMN,
                    AIRCRAFT_TYPE'image(
                        ITEM.AIRCRAFT_KIND_VALUE));
            when LAUNCHER =>
                if ITEM.LAUNCHER_VALUE /=
                LAUNCHER_TYPE'last then
                    ITEM.LAUNCHER_VALUE :=
                    LAUNCHER_TYPE'succ(
                        ITEM.LAUNCHER_VALUE);
                else
                    ITEM.LAUNCHER_VALUE :=
                    LAUNCHER_TYPE'first;
                end if;
                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                    DATA_COLUMN, BLANK_STRING(
                        1..LAUNCHER_TYPE'width));
                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                    DATA_COLUMN,
                    LAUNCHER_TYPE'image(
                        ITEM.LAUNCHER_VALUE));
            when SSJ_ECM =>
                if ITEM.SSJ_ECM_VALUE /=
                SSJ_ECM_TYPE'last then
                    ITEM.SSJ_ECM_VALUE :=
                    SSJ_ECM_TYPE'succ(
                        ITEM.SSJ_ECM_VALUE);
                else
                    ITEM.SSJ_ECM_VALUE :=
                    SSJ_ECM_TYPE'first;
                end if;
                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                    DATA_COLUMN, BLANK_STRING(
                        1..SSJ_ECM_TYPE'width));
                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                    DATA_COLUMN,
                    SSJ_ECM_TYPE'image(
                        ITEM.SSJ_ECM_VALUE));
            when SOJ_ECM =>
                if ITEM.SOJ_ECM_VALUE /=
                SOJ_ECM_TYPE'last then
                    ITEM.SOJ_ECM_VALUE :=
                    SOJ_ECM_TYPE'succ(
                        ITEM.SOJ_ECM_VALUE);
                else
                    ITEM.SOJ_ECM_VALUE :=
                    SOJ_ECM_TYPE'first;
                end if;
                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                    DATA_COLUMN, BLANK_STRING(
                        1..SOJ_ECM_TYPE'width));
                SYSTEM_SPECIFIC.PUT_STRING( ROW,
                    DATA_COLUMN,
                    SOJ_ECM_TYPE'image(
                        ITEM.SOJ_ECM_VALUE));
        end case;
    when ACTION =>
        case ITEM.ACTION_KIND is
            when LOAD_DATA =>
                STATUS_OK := LOAD_DATA_FILE;
                if STATUS_OK then
                    NUM_ITEMS := DISPLAY_MENU(MENU,
                        ITEM_NUM);
                    SYSTEM_SPECIFIC.PUT_STRING(22, 10,
                        "File loaded successfully");
                    MESSAGE_DISPLAYED := true;
                end if;
            when SAVE_DATA =>
                STATUS_OK := SAVE_DATA_FILE;
                if STATUS_OK then
                    SYSTEM_SPECIFIC.PUT_STRING(22, 10,
                    "File saved successfully");
                MESSAGE_DISPLAYED := true;
            end if;
        when START_RUN =>
            START_RUN;
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP1_GRAPH1 => -- Elev. View of Traj
              PLOT.DO_GRAPH(21);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP1_GRAPH2 => -- Plan View of Traj
              PLOT.DO_GRAPH(22);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP1_GRAPH3 => -- Msl-Tgt Range Rate
              PLOT.DO_GRAPH(7);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP1_GRAPH4 => -- Msl-Tgt Range
              PLOT.DO_GRAPH(6);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP1_GRAPH5 => -- LOS Angle Rates
              PLOT.DO_GRAPH(10);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP1_GRAPH6 => -- Tgt Heading Angle
              PLOT.DO_GRAPH(25);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP1_GRAPH7 => -- Launch A/C-Tgt Range
              PLOT.DO_GRAPH(1);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP2_GRAPH1 => -- Msl. Mach
              PLOT.DO_GRAPH(5);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP2_GRAPH2 => -- Msl. Velocity
              PLOT.DO_GRAPH(2);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP2_GRAPH3 => -- Msl. Alt. Rate
              PLOT.DO_GRAPH(11);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP2_GRAPH4 => -- Msl. Pitch Angle
              PLOT.DO_GRAPH(4);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP2_GRAPH5 => -- Msl. Yaw Angle
              PLOT.DO_GRAPH(3);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP3_GRAPH1 => -- Msl. Thrust
              PLOT.DO_GRAPH(16);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP3_GRAPH2 => -- Msl. Fuel Mass
              PLOT.DO_GRAPH(14);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP3_GRAPH3 => -- Msl. Fuel Rate
              PLOT.DO_GRAPH(17);
            NUM_ITEMS := DISPLAY_MENU(MENU,
                ITEM_NUM);
        when GRP4_GRAPH1 => -- Msl. Acc. Cmds.
              PLOT.DO_GRAPH(12);
```

```
                                        NUM_ITEMS := DISPLAY_MENU(MENU,
                                            ITEM_NUM);
                                    when GRP4_GRAPH2 => -- Msl. Ach. Acc.
                                         PLOT.DO_GRAPH(13);
                                        NUM_ITEMS := DISPLAY_MENU(MENU,
                                            ITEM_NUM);
                                    when GRP4_GRAPH3 => -- Msl. Alpha
                                         PLOT.DO_GRAPH(18);
                                        NUM_ITEMS := DISPLAY_MENU(MENU,
                                            ITEM_NUM);
                                    when GRP4_GRAPH4 => -- Msl. Weight
                                         PLOT.DO_GRAPH(15);
                                        NUM_ITEMS := DISPLAY_MENU(MENU,
                                            ITEM_NUM);
                                    when GRP4_GRAPH5 => -- Msl. Skr. Angles
                                         PLOT.DO_GRAPH(9);
                                        NUM_ITEMS := DISPLAY_MENU(MENU,
                                            ITEM_NUM);
                                    when GRP4_GRAPH6 => -- Msl. RF SNR
                                         PLOT.DO_GRAPH(19);
                                        NUM_ITEMS := DISPLAY_MENU(MENU,
                                            ITEM_NUM);
                                    when GRP4_GRAPH7 => -- Msl. Drag Coef.
                                         PLOT.DO_GRAPH(20);
                                        NUM_ITEMS := DISPLAY_MENU(MENU,
                                            ITEM_NUM);
                                end case;
                        end case;
                    end if;
                when others =>
                    null;
            end case;
        end loop;
    end MANAGE_MENU;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure MAIN is
    begin
        MANAGE_MENU(MAIN_MENU);
    end MAIN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure DRAW_RUNTIME_BORDER is
    begin
        SYSTEM_SPECIFIC.DRAW_BOX;

        SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
        SYSTEM_SPECIFIC.PUT_STRING( 0, 19, MISSILE.RUNTIME_TITLE);
        SYSTEM_SPECIFIC.PUT_STRING(24, 25," Press space bar to pause run ");
        SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
    end DRAW_RUNTIME_BORDER;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SETUP_RUNTIME_SCREEN is separate;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SETUP_MENU_DATA is separate;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    task body KEYBOARD_HANDLER is separate;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
begin
    SETUP_MENU_DATA;
end USER_INTERFACE;
--------------------------------------------------------------------------------
-- User Interface Package Subunit
--
-- This subunit contains all the data describing the user interface.
```

```
--
--------------------------------------------------------------------------------

separate (USER_INTERFACE)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure SETUP_MENU_DATA is
begin

    -- Graph Group 4 Main Menu

    GRAPH_GRP4_MENU_ITEMS(7) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Drag Coefficient          ",
        DATA_TYPE => NONE, NEXT_ITEM => null, ACTION_KIND => GRP4_GRAPH7);

    GRAPH_GRP4_MENU_ITEMS(6) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl RF S/N Ratio              ",
        DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP4_MENU_ITEMS(7),
        ACTION_KIND => GRP4_GRAPH6);

    GRAPH_GRP4_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Seeker Gimbal Angles      ",
        DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP4_MENU_ITEMS(6),
        ACTION_KIND => GRP4_GRAPH5);

    GRAPH_GRP4_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Weight                    ",
        DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP4_MENU_ITEMS(5),
        ACTION_KIND => GRP4_GRAPH4);

    GRAPH_GRP4_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Angle of Attack           ",
        DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP4_MENU_ITEMS(4),
        ACTION_KIND => GRP4_GRAPH3);

    GRAPH_GRP4_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Achieved Acceleration     ",
        DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP4_MENU_ITEMS(3),
        ACTION_KIND => GRP4_GRAPH2);

    GRAPH_GRP4_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Acceleration Commands     ",
        DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP4_MENU_ITEMS(2),
        ACTION_KIND => GRP4_GRAPH1);

    GRAPH_GRP4_MENU := new MENU_TYPE'(TITLE =>
        "                  Msl Parameter Graph Menu                  ",
        FIRST_ITEM => GRAPH_GRP4_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

    -- Graph Group 3 Main Menu

    GRAPH_GRP3_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Fuel Flow Rate            ",
        DATA_TYPE => NONE, NEXT_ITEM => null, ACTION_KIND => GRP3_GRAPH3);

    GRAPH_GRP3_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Fuel Weight               ",
        DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP3_MENU_ITEMS(3),
        ACTION_KIND => GRP3_GRAPH2);

    GRAPH_GRP3_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Msl Thrust                    ",
        DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP3_MENU_ITEMS(2),
        ACTION_KIND => GRP3_GRAPH1);

    GRAPH_GRP3_MENU := new MENU_TYPE'(TITLE =>
        "                 Msl Propulsion Graph Menu                  ",
        FIRST_ITEM => GRAPH_GRP3_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);
```

```
-- Graph Group 2 Main Menu

GRAPH_GRP2_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Msl Yaw Angle                    ",
    DATA_TYPE => NONE, NEXT_ITEM => null, ACTION_KIND => GRP2_GRAPH5);

GRAPH_GRP2_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Msl Pitch Angle                  ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP2_MENU_ITEMS(5),
    ACTION_KIND => GRP2_GRAPH4);

GRAPH_GRP2_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Msl Altitude Rate                ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP2_MENU_ITEMS(4),
    ACTION_KIND => GRP2_GRAPH3);

GRAPH_GRP2_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Msl Velocity                     ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP2_MENU_ITEMS(3),
    ACTION_KIND => GRP2_GRAPH2);

GRAPH_GRP2_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Msl Mach Number                  ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP2_MENU_ITEMS(2),
    ACTION_KIND => GRP2_GRAPH1);

GRAPH_GRP2_MENU := new MENU_TYPE'(TITLE =>
    "                Msl Kinematics Graph Menu                  ",
    FIRST_ITEM => GRAPH_GRP2_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

-- Graph Group 1 Main Menu

GRAPH_GRP1_MENU_ITEMS(7) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Launch A/C-Tgt Range             ",
    DATA_TYPE => NONE, NEXT_ITEM => null, ACTION_KIND => GRP1_GRAPH7);

GRAPH_GRP1_MENU_ITEMS(6) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Tgt Heading Angle                ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP1_MENU_ITEMS(7),
    ACTION_KIND => GRP1_GRAPH6);

GRAPH_GRP1_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Msl-Tgt LOS Angle Rates          ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP1_MENU_ITEMS(6),
    ACTION_KIND => GRP1_GRAPH5);

GRAPH_GRP1_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Msl-Tgt Range                    ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP1_MENU_ITEMS(5),
    ACTION_KIND => GRP1_GRAPH4);

GRAPH_GRP1_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Msl-Tgt Range Rate               ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP1_MENU_ITEMS(4),
    ACTION_KIND => GRP1_GRAPH3);

GRAPH_GRP1_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Plan view of trajectory          ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP1_MENU_ITEMS(3),
    ACTION_KIND => GRP1_GRAPH2);

GRAPH_GRP1_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
    TITLE => "Elevation view of trajectory     ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_GRP1_MENU_ITEMS(2),
    ACTION_KIND => GRP1_GRAPH1);

GRAPH_GRP1_MENU := new MENU_TYPE'(TITLE =>
    "               Msl-Tgt Geometry Graph Menu                 ",
    FIRST_ITEM => GRAPH_GRP1_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);
```

```
-- Graph Main Menu

GRAPH_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
    "Msl Parameter Graph Menu       ",
    DATA_TYPE => NONE, NEXT_ITEM => null, NEXT_MENU => GRAPH_GRP4_MENU);

GRAPH_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
    "Msl Propulsion Graph Menu      ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_MENU_ITEMS(4), NEXT_MENU =>
    GRAPH_GRP3_MENU);

GRAPH_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
    "Msl Kinematics Graph Menu      ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_MENU_ITEMS(3), NEXT_MENU =>
    GRAPH_GRP2_MENU);

GRAPH_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
    "Msl-Tgt Geometry Graph Menu    ",
    DATA_TYPE => NONE, NEXT_ITEM => GRAPH_MENU_ITEMS(2), NEXT_MENU =>
    GRAPH_GRP1_MENU);

GRAPH_MENU := new MENU_TYPE'(TITLE =>
    "                         Graph Menus                         ",
    FIRST_ITEM => GRAPH_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

-- SOJ 2 Parameter Menu

SOJ2_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE =>  "SOJ ERPD, dBW/MHz:              ",
    PROMPT => "Enter SOJ ERPD (dBW/MHz):      ",
    REAL_VALUE => 30.0, MAX_VALUE => 200.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => null);

SOJ2_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "SOJ ECM technique:              ",
    SOJ_ECM_VALUE => BARRAGE_NOISE,
    PROMPT => "                               ",
    DATA_TYPE => SOJ_ECM, NEXT_ITEM => SOJ2_MENU_ITEMS(5));

SOJ2_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Look angle rel. to LAC, deg:  ",
    PROMPT => "Enter angle (deg)(+right):     ",
    REAL_VALUE => 0.0, MAX_VALUE => 90.0, MIN_VALUE => -90.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => SOJ2_MENU_ITEMS(4));

SOJ2_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Range from launch A/C, NMI:    ",
    PROMPT => "Enter distance (NMI):          ",
    REAL_VALUE => 150.0, MAX_VALUE => 500.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => SOJ2_MENU_ITEMS(3));

SOJ2_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "SOJ altitude, kft:              ",
    PROMPT => "Enter altitude (kft):          ",
    REAL_VALUE => 30.0, MAX_VALUE => 150.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => SOJ2_MENU_ITEMS(2));

SOJ2_MENU := new MENU_TYPE'(TITLE =>
    "                     SOJ 2 Parameter Menu                     ",
    FIRST_ITEM => SOJ2_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

-- SOJ 1 Parameter Menu

SOJ1_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "SOJ ERPD, dBW/MHz:              ",
    PROMPT => "Enter SOJ ERPD (dBW/MHz):      ",
    REAL_VALUE => 30.0, MAX_VALUE => 200.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => null);
```

```
SOJ1_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "SOJ ECM technique:               ",
    SOJ_ECM_VALUE => BARRAGE_NOISE,
    PROMPT => "                                ",
    DATA_TYPE => SOJ_ECM, NEXT_ITEM => SOJ1_MENU_ITEMS(5));

SOJ1_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Look angle rel. to LAC, deg:  ",
    PROMPT => "Enter angle (deg)(+right):    ",
    REAL_VALUE => 0.0, MAX_VALUE => 90.0, MIN_VALUE => -90.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => SOJ1_MENU_ITEMS(4));

SOJ1_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Range from launch A/C, NMI:   ",
    PROMPT => "Enter distance (NMI):          ",
    REAL_VALUE => 150.0, MAX_VALUE => 500.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => SOJ1_MENU_ITEMS(3));

SOJ1_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "SOJ altitude, kft:              ",
    PROMPT => "Enter altitude (kft):          ",
    REAL_VALUE => 30.0, MAX_VALUE => 150.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => SOJ1_MENU_ITEMS(2));

SOJ1_MENU := new MENU_TYPE'(TITLE =>
    "                     SOJ 1 Parameter Menu                    ",
    FIRST_ITEM => SOJ1_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

-- Target 2 Parameter Menu

TGT2_MENU_ITEMS(7) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "ERPD, dBW/Mhz; LG, dB:          ",
    PROMPT => "Enter ERPD/LG (dBW/Mhz:dB):   ",
    REAL_VALUE => 30.0, MAX_VALUE => 200.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => null);

TGT2_MENU_ITEMS(6) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "ECM technique:                  ",
    SSJ_ECM_VALUE => NONE,
    PROMPT => "                                ",
    DATA_TYPE => SSJ_ECM, NEXT_ITEM => TGT2_MENU_ITEMS(7));

TGT2_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target IR radiance:             ",
    TGT_IR_SIZE_VALUE => MEDIUM,
    PROMPT => "                                ",
    DATA_TYPE => TGT_IR_SIZE, NEXT_ITEM => TGT2_MENU_ITEMS(6));

TGT2_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target RCS, square meters:    ",
    PROMPT => "Enter RCS (square meters):    ",
    REAL_VALUE => 2.0, MAX_VALUE => 100.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT2_MENU_ITEMS(5));

TGT2_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Echelon angle, deg (90 trail):",
    PROMPT => "Enter echelon angle, (deg):   ",
    REAL_VALUE => 0.0, MAX_VALUE => 180.0, MIN_VALUE => -180.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT2_MENU_ITEMS(4));

TGT2_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Range to target 1, feet:       ",
    PROMPT => "Enter range (feet):            ",
    REAL_VALUE => 100.0, MAX_VALUE => 100000.0, MIN_VALUE => -100000.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT2_MENU_ITEMS(3));

TGT2_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target altitude, kft:           ",
    PROMPT => "Enter altitude (kft):          ",
    REAL_VALUE => 30.0, MAX_VALUE => 150.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT2_MENU_ITEMS(2));

TGT2_MENU := new MENU_TYPE'(TITLE =>
    "                   Target 2 Parameter Menu                  ",
    FIRST_ITEM => TGT2_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

-- Target 1 Parameter Menu

TGT1_MENU_ITEMS(15) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "ERPD, dBW/Mhz; Loop gain, dB: ",
    PROMPT => "Enter ERPD/LG (dBW/Mhz:dB)  : ",
    REAL_VALUE => 30.0, MAX_VALUE => 200.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => null);

TGT1_MENU_ITEMS(14) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "ECM technique:                ",
    SSJ_ECM_VALUE => NONE,
    PROMPT => "                              ",
    DATA_TYPE => SSJ_ECM, NEXT_ITEM => TGT1_MENU_ITEMS(15));

TGT1_MENU_ITEMS(13) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Weave period, sec:            ",
    PROMPT => "Enter weave period (sec):     ",
    REAL_VALUE => 20.0, MAX_VALUE => 100.0, MIN_VALUE => 1.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(14));

TGT1_MENU_ITEMS(12) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Angle to turn, deg (+right):  ",
    PROMPT => "Enter angle to turn (deg):    ",
    REAL_VALUE => 10.0, MAX_VALUE => 3600.0, MIN_VALUE => -3600.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(13));

TGT1_MENU_ITEMS(11) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Maneuver buildup time, sec:   ",
    PROMPT => "Enter buildup time (sec):     ",
    REAL_VALUE => 1.0, MAX_VALUE => 10.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(12));

TGT1_MENU_ITEMS(10) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Man. start value, sec or NMI: ",
    PROMPT => "Enter start value (sec / NMI):",
    REAL_VALUE => 10.0, MAX_VALUE => 500.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(11));

TGT1_MENU_ITEMS(9) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Maneuver start parameter:     ",
    MANEUVER_START_VALUE => FLIGHT_TIME,
    PROMPT => "                              ",
    DATA_TYPE => MANEUVER_START, NEXT_ITEM => TGT1_MENU_ITEMS(10));

TGT1_MENU_ITEMS(8) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Maneuver g's:                 ",
    PROMPT => "Enter maneuver g's:           ",
    REAL_VALUE => 2.0, MAX_VALUE => 12.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(9));

TGT1_MENU_ITEMS(7) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Maneuver type:                ",
    MANEUVER_VALUE => NONE,
    PROMPT => "                              ",
    DATA_TYPE => MANEUVER, NEXT_ITEM => TGT1_MENU_ITEMS(8));

TGT1_MENU_ITEMS(6) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target IR radiance:           ",
    TGT_IR_SIZE_VALUE => MEDIUM,
    PROMPT => "                              ",
    DATA_TYPE => TGT_IR_SIZE, NEXT_ITEM => TGT1_MENU_ITEMS(7));
```

```
TGT1_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target RCS, square meters:    ",
    PROMPT => "Enter RCS (square meters):    ",
    REAL_VALUE => 2.0, MAX_VALUE => 100.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(6));

TGT1_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target slant range, NMI:      ",
    PROMPT => "Enter range (NMI):            ",
    REAL_VALUE => 20.0, MAX_VALUE => 500.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(5));

TGT1_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target aspect angle, deg:     ",
    PROMPT => "Enter aspect angle (deg):     ",
    REAL_VALUE => 180.0, MAX_VALUE => 360.0, MIN_VALUE => -360.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(4));

TGT1_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target speed, mach:           ",
    PROMPT => "Enter mach number:            ",
    REAL_VALUE => 0.9, MAX_VALUE => 4.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(3));

TGT1_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Target altitude, kft:         ",
    PROMPT => "Enter altitude (kft):         ",
    REAL_VALUE => 30.0, MAX_VALUE => 150.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => TGT1_MENU_ITEMS(2));

TGT1_MENU := new MENU_TYPE'(TITLE =>
    "                    Target 1 Parameter Menu                   ",
    FIRST_ITEM => TGT1_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 1);

-- Target Parameter Menus

TGT_MENU_ITEMS(7) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
    "SOJ 2 Parameter Menu          ",
    DATA_TYPE => NONE, NEXT_ITEM => null, NEXT_MENU =>
    SOJ2_MENU);

TGT_MENU_ITEMS(6) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Enable SOJ 2:                 ",
    YES_NO_VALUE => NO, PROMPT => "                              ",
    DATA_TYPE => YES_NO, NEXT_ITEM => TGT_MENU_ITEMS(7));

TGT_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
    "SOJ 1 Parameter Menu          ",
    DATA_TYPE => NONE, NEXT_ITEM => TGT_MENU_ITEMS(6), NEXT_MENU =>
    SOJ1_MENU);

TGT_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Enable SOJ 1:                 ",
    YES_NO_VALUE => NO, PROMPT => "                              ",
    DATA_TYPE => YES_NO, NEXT_ITEM => TGT_MENU_ITEMS(5));

TGT_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
    "Target 2 Parameter Menu       ",
    DATA_TYPE => NONE, NEXT_ITEM => TGT_MENU_ITEMS(4), NEXT_MENU =>
    TGT2_MENU);

TGT_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Enable target 2:              ",
    YES_NO_VALUE => NO, PROMPT => "                              ",
    DATA_TYPE => YES_NO, NEXT_ITEM => TGT_MENU_ITEMS(3));

TGT_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
    "Target 1 Parameter Menu       ",
    DATA_TYPE => NONE, NEXT_ITEM => TGT_MENU_ITEMS(2), NEXT_MENU =>
    TGT1_MENU);

TGT_MENU := new MENU_TYPE'(TITLE =>
    "                    Target Parameter Menus                  ",
    FIRST_ITEM => TGT_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

-- Launch Aircraft Parameter Menu

LAUNCHER_MENU_ITEMS(6) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Launch A/C guidance mode:     ",
    INT_GUIDANCE_VALUE => NON_MANEUVERING,
    PROMPT => "                             ",
    DATA_TYPE => INT_GUIDANCE, NEXT_ITEM => null);

LAUNCHER_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Launch A/C lead angle, deg:   ",
    PROMPT => "Enter lead angle (deg):      ",
    REAL_VALUE => 0.0, MAX_VALUE => 90.0, MIN_VALUE => -90.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => LAUNCHER_MENU_ITEMS(6));

LAUNCHER_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Launch A/C speed, mach:       ",
    PROMPT => "Enter mach number:           ",
    REAL_VALUE => 0.9, MAX_VALUE => 3.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => LAUNCHER_MENU_ITEMS(5));

LAUNCHER_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Launch A/C altitude, kft:     ",
    PROMPT => "Enter altitude (kft):        ",
    REAL_VALUE => 30.0, MAX_VALUE => 70.0, MIN_VALUE => 0.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => LAUNCHER_MENU_ITEMS(4));

LAUNCHER_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Launcher type:                ",
    LAUNCHER_VALUE => RAIL, PROMPT => "                             ",
    DATA_TYPE => LAUNCHER, NEXT_ITEM => LAUNCHER_MENU_ITEMS(3));

LAUNCHER_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Launch A/C type:              ",
    AIRCRAFT_KIND_VALUE => F_14, PROMPT => "                             ",
    DATA_TYPE => AIRCRAFT_KIND, NEXT_ITEM => LAUNCHER_MENU_ITEMS(2));

LAUNCHER_MENU := new MENU_TYPE'(TITLE =>
    "                Launch Aircraft Parameter Menu              ",
    FIRST_ITEM => LAUNCHER_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

-- File Operations Menu

FILE_MENU_ITEMS(7) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Data log interval, frames:    ",
    PROMPT => "Enter log interval (frames):  ",
    REAL_VALUE => 4.0, MAX_VALUE => 100.0, MIN_VALUE => 1.0,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => null);

FILE_MENU_ITEMS(6) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Simulation frame time, sec:   ",
    PROMPT => "Enter frame time (sec):       ",
    REAL_VALUE => 0.25, MAX_VALUE => 10.0, MIN_VALUE => 0.01,
    DATA_TYPE => FLOATING_PT, NEXT_ITEM => FILE_MENU_ITEMS(7));

FILE_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Write output data to file:    ",
    YES_NO_VALUE => NO, PROMPT => "                             ",
    DATA_TYPE => YES_NO, NEXT_ITEM => FILE_MENU_ITEMS(6));

FILE_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
    TITLE => "Name of output data file:     ",
    PROMPT => "Enter name of output file:    ",
    TEXT_VALUE => MISSILE.OUTFILE,
        DATA_TYPE => TEXT, NEXT_ITEM => FILE_MENU_ITEMS(5));

    FILE_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Save scenario data file        ",
        DATA_TYPE => NONE, NEXT_ITEM => FILE_MENU_ITEMS(4),
        ACTION_KIND => SAVE_DATA);

    FILE_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION,
        TITLE => "Load scenario data file        ",
        DATA_TYPE => NONE, NEXT_ITEM => FILE_MENU_ITEMS(3),
        ACTION_KIND => LOAD_DATA);

    FILE_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => DATA_ITEM,
        TITLE => "Name of scenario data file:    ",
        PROMPT => "Enter name of scenario file:  ",
        TEXT_VALUE => MISSILE.DATFILE,
        DATA_TYPE => TEXT, NEXT_ITEM => FILE_MENU_ITEMS(2));

    FILE_MENU := new MENU_TYPE'(TITLE =>
        "                    File Operations Menu                    ",
        FIRST_ITEM => FILE_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

    -- Main Menu

    MAIN_MENU_ITEMS(5) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
        "Graph Data from Previous Run  ",
        DATA_TYPE => NONE, NEXT_ITEM => null, NEXT_MENU => GRAPH_MENU);

    MAIN_MENU_ITEMS(4) := new MENU_ITEM_TYPE'(ITEM_KIND => ACTION, TITLE =>
        "Start Simulation Run          ",
        DATA_TYPE => NONE, NEXT_ITEM => MAIN_MENU_ITEMS(5),
        ACTION_KIND => START_RUN);

    MAIN_MENU_ITEMS(3) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
        "Target Parameter Menus        ",
        DATA_TYPE => NONE, NEXT_ITEM => MAIN_MENU_ITEMS(4), NEXT_MENU =>
        TGT_MENU);

    MAIN_MENU_ITEMS(2) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
        "Launch Aircraft Parameter Menu",
        DATA_TYPE => NONE, NEXT_ITEM => MAIN_MENU_ITEMS(3), NEXT_MENU =>
        LAUNCHER_MENU);

    MAIN_MENU_ITEMS(1) := new MENU_ITEM_TYPE'(ITEM_KIND => SUBMENU, TITLE =>
        "File Operations Menu          ",
        DATA_TYPE => NONE, NEXT_ITEM => MAIN_MENU_ITEMS(2), NEXT_MENU =>
        FILE_MENU);

    MAIN_MENU := new MENU_TYPE'(TITLE => MISSILE.MAIN_MENU_TITLE,
        FIRST_ITEM => MAIN_MENU_ITEMS(1), ROWS_BETWEEN_ITEMS => 2);

end SETUP_MENU_DATA;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
separate (USER_INTERFACE)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure SETUP_RUNTIME_SCREEN is
begin
    SYSTEM_SPECIFIC.CLEAR_SCREEN;
    DRAW_RUNTIME_BORDER;

    SYSTEM_SPECIFIC.PUT_STRING( 1, 3, "Elapsed Time, sec         :");
    SYSTEM_SPECIFIC.PUT_STRING( 2, 3, "Flight Time, sec          :");
    SYSTEM_SPECIFIC.PUT_STRING( 3, 3, "Msl Time to Go, sec       :");
    SYSTEM_SPECIFIC.PUT_STRING( 4, 3,
        "-----------------------------------");
    SYSTEM_SPECIFIC.PUT_STRING( 5, 3, "Msl RF Seeker Mode        :");
    SYSTEM_SPECIFIC.PUT_STRING( 6, 3, "Msl IR Seeker Mode        :");
    SYSTEM_SPECIFIC.PUT_STRING( 7, 3, "Msl Az Gimbal Angle, deg :");
    SYSTEM_SPECIFIC.PUT_STRING( 8, 3, "Msl El Gimbal Angle, deg :");
    SYSTEM_SPECIFIC.PUT_STRING( 9, 3, "Msl Az LOS Rate, deg/sec :");
    SYSTEM_SPECIFIC.PUT_STRING(10, 3, "Msl El LOS Rate, deg/sec :");
    SYSTEM_SPECIFIC.PUT_STRING(11, 3,
        "------------------------------------");
    SYSTEM_SPECIFIC.PUT_STRING(12, 3, "Msl Guidance Mode        :");
    SYSTEM_SPECIFIC.PUT_STRING(13, 3, "Msl Az Accel Cmd, g's    :");
    SYSTEM_SPECIFIC.PUT_STRING(14, 3, "Msl El Accel Cmd, g's    :");
    SYSTEM_SPECIFIC.PUT_STRING(15, 3,
        "------------------------------------");
    SYSTEM_SPECIFIC.PUT_STRING(16, 3, "Msl Propulsion Mode      :");
    SYSTEM_SPECIFIC.PUT_STRING(17, 3, "Msl Thrust, lbs          :");
    SYSTEM_SPECIFIC.PUT_STRING(18, 3,
        "-----------------------------------  ");
    SYSTEM_SPECIFIC.PUT_STRING(19, 3, "Msl Weight, lbs          :");
    SYSTEM_SPECIFIC.PUT_STRING(20, 3, "Msl Angle of Attack, deg :");
    SYSTEM_SPECIFIC.PUT_STRING(21, 3, "Msl Sideslip Angle, deg  :");
    SYSTEM_SPECIFIC.PUT_STRING(22, 3,
        "-----------------------------------  ");
    SYSTEM_SPECIFIC.PUT_STRING( 1,41, "Msl Mach                 :");
    SYSTEM_SPECIFIC.PUT_STRING( 2,41, "Msl Velocity, ft/sec     :");
    SYSTEM_SPECIFIC.PUT_STRING( 3,41, "Msl Altitude, ft         :");
    SYSTEM_SPECIFIC.PUT_STRING( 4,41, "Msl Altitude Rate, ft/sec:");
    SYSTEM_SPECIFIC.PUT_STRING( 5,41, "Msl Pitch Angle, deg     :");
    SYSTEM_SPECIFIC.PUT_STRING( 6,41, "Msl Heading Angle, deg   :");
    SYSTEM_SPECIFIC.PUT_STRING( 7,41, "Msl Downrange Pos, NMI   :");
    SYSTEM_SPECIFIC.PUT_STRING( 8,41, "Msl Crossrange Pos, NMI  :");
    SYSTEM_SPECIFIC.PUT_STRING( 9,41,
        "-------------------------------------");
    SYSTEM_SPECIFIC.PUT_STRING(10,41, "Msl X Axis Accel, g's    :");
    SYSTEM_SPECIFIC.PUT_STRING(11,41, "Msl Y Axis Accel, g's    :");
    SYSTEM_SPECIFIC.PUT_STRING(12,41, "Msl Z Axis Accel, g's    :");
    SYSTEM_SPECIFIC.PUT_STRING(13,41,
        "-------------------------------------");
    SYSTEM_SPECIFIC.PUT_STRING(14,41, "Launcher-Tgt Range, NMI  :");
    SYSTEM_SPECIFIC.PUT_STRING(15,41, "Msl-Tgt Range, NMI       :");
    SYSTEM_SPECIFIC.PUT_STRING(16,41, "Msl-Tgt Range Rate,ft/sec:");
    SYSTEM_SPECIFIC.PUT_STRING(17,41, "Tgt Mach                 :");
    SYSTEM_SPECIFIC.PUT_STRING(18,41, "Tgt Altitude, ft         :");
    SYSTEM_SPECIFIC.PUT_STRING(19,41, "Tgt Heading Angle, deg   :");
    SYSTEM_SPECIFIC.PUT_STRING(20,41,
        "-------------------------------------");
end SETUP_RUNTIME_SCREEN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
separate (USER_INTERFACE)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure SHOW_TITLE_SCREEN is
    NUM_TITLE_SCREEN_LINES : constant := 21;
    TITLE_SCREEN_TEXT : STRING_ARRAY(1..NUM_TITLE_SCREEN_LINES) := (
        "     NN     NN        PPPPPPPPPP        SSSSSSSSSS           ",
        "     NNN    NN        PP      PP        SS                   ",
        "     NN N   NN        PP      PP        SS                   ",
        "     NN  N  NN        PPPPPPPPPP        SSSSSSSSSS           ",
        "     NN   N NN        PP                        SS           ",
        "     NN    NNN        PP                        SS           ",
        "     NN     NN        PP                SSSSSSSSSS           ",
        "                                                            ",
        MISSILE.TITLE_SCREEN_LINE,
        "                                                            ",
        "                         HELLO!                             ",
        "                                                            ",
        "The simulation represents a generic air-to-air missile and  ",
        "there for is                                                ",
        "                                                            ",
        "                       UNCLASSIFIED!                        ",
        "                                                            ",
        "                          ENJOY!                          ",
        "                                                          ",
        "                                                          ",
        "                                                          ");

    CHAR  : character;
    ROW   : integer;
begin
    SYSTEM_SPECIFIC.CLEAR_SCREEN;
    SYSTEM_SPECIFIC.TURN_CURSOR_OFF;
    SYSTEM_SPECIFIC.DRAW_BOX;

    ROW := 2;
    for I in 1..NUM_TITLE_SCREEN_LINES loop
        SYSTEM_SPECIFIC.PUT_STRING(ROW, 10, TITLE_SCREEN_TEXT(I));
        ROW := ROW + 1;
    end loop;

    SYSTEM_SPECIFIC.REVERSE_VIDEO_ON;
    SYSTEM_SPECIFIC.PUT_STRING(24,25, " Press the space bar to begin ");
    SYSTEM_SPECIFIC.REVERSE_VIDEO_OFF;
    KEYBOARD_HANDLER.GET_KEY_WAIT(CHAR);
end SHOW_TITLE_SCREEN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
separate (USER_INTERFACE)
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
task body KEYBOARD_HANDLER is
    BUFFER_SIZE : constant := 100;
    BUFFER : array(1..BUFFER_SIZE) of character;
    HEAD_INDEX : integer := 1;
    TAIL_INDEX : integer := 1;
    KEYS_IN_BUFFER : integer := 0;

begin
    loop
        while SYSTEM_SPECIFIC.KEY_AVAILABLE and
        KEYS_IN_BUFFER < BUFFER_SIZE loop

            BUFFER(HEAD_INDEX) := SYSTEM_SPECIFIC.GET_KEY;
            HEAD_INDEX := (HEAD_INDEX mod BUFFER_SIZE) + 1;
            KEYS_IN_BUFFER := KEYS_IN_BUFFER + 1;
        end loop;

        select
            accept KEY_AVAILABLE(KEY_IN_BUFFER : out boolean) do
                if KEYS_IN_BUFFER > 0 then
                    KEY_IN_BUFFER := true;
                else
                    KEY_IN_BUFFER := false;
                end if;
            end KEY_AVAILABLE;
        or
            when KEYS_IN_BUFFER > 0 =>
            accept GET_KEY_WAIT(KEY : out character) do

                KEY := BUFFER(TAIL_INDEX);
                TAIL_INDEX := (TAIL_INDEX mod BUFFER_SIZE) + 1;
                KEYS_IN_BUFFER := KEYS_IN_BUFFER - 1;
            end GET_KEY_WAIT;
        or
            accept GET_KEY_NOWAIT(KEY : out character;
                KEY_VALID : out boolean) do

                if KEYS_IN_BUFFER > 0 then
                    KEY := BUFFER(TAIL_INDEX);
                    TAIL_INDEX := (TAIL_INDEX mod BUFFER_SIZE) + 1;
                    KEYS_IN_BUFFER := KEYS_IN_BUFFER - 1;
                    KEY_VALID := true;
                else
                    KEY_VALID := false;
                end if;
            end GET_KEY_NOWAIT;
        else
            null;
        end select;
    end loop;
end KEYBOARD_HANDLER;
--------------------------------------------------------------------------------
-- System Specific Package Specification
--
-- This package contains the routines that are specific to the particular
-- computer system that it is running on.
--
--------------------------------------------------------------------------------

with MATH; use MATH;

package SYSTEM_SPECIFIC is
    type MENU_COMMAND is (
        UP_ARROW,
        DOWN_ARROW,
        ENTER_KEY);

    procedure INIT_VIDEO;

    function IS_MONOCHROME
    return boolean;

    procedure CLEAR_SCREEN;

    procedure REVERSE_VIDEO_ON;

    procedure REVERSE_VIDEO_OFF;

    procedure MOVE_CURSOR (
        ROW : integer;
        COLUMN : integer);

    procedure TURN_CURSOR_ON;

    procedure TURN_CURSOR_OFF;

    procedure PUT_STRING (
        SCREEN_ROW : integer;
        SCREEN_COLUMN : integer;
        TEXT_STRING : string);

    procedure PUT_REAL (
        ROW : integer;
        COLUMN : integer;
        NUMBER : REAL;
        AFTER : integer := 2);

    procedure INPUT_STRING (
        TEXT_STRING : out string);

    function GET_KEY
    return character;

    function KEY_AVAILABLE
    return boolean;

    function GET_MENU_COMMAND
    return MENU_COMMAND;

    procedure DRAW_BOX;
    procedure QUIT_PROGRAM;

end SYSTEM_SPECIFIC;
-------------------------------------------------------------------------------
-- System Specific Package Body
--
-- This package contains the routines that are specific to the particular
-- computer system that it is running on.
--
-------------------------------------------------------------------------------

with MATH; use MATH;
with BIT_OPS; use BIT_OPS;
with COMMON_DISPLAY_TYPES;
with System;
with PROGRAM_CONTROL;
with INTERRUPT;
with TEXT_IO;
with REAL_IO;
with TTY;
with BOX;
with EQUIPMENT;

package body SYSTEM_SPECIFIC is
    MONOCHROME  : boolean;
    REVERSE_VIDEO : boolean := false;
    FAST_VIDEO  : boolean := true;
    EGA_FLAG  : boolean;
    COPROCESSOR_FLAG  : boolean;

    -- Set up video buffer as an array

    MONO_BUFFER_ADDR : System.Address := System.Address(16#B000_0000#);
    COLOR_BUFFER_ADDR : System.Address := System.Address(16#B800_0000#);

    type byte is range 0..16#FF#;

    type SCREEN_CHAR_TYPE is
    record
        CHAR : byte;
        ATTRIBUTE : byte;
    end record;

    pragma PACK(SCREEN_CHAR_TYPE);

    MONO_BUFFER  : array (0..24, 0..79) of SCREEN_CHAR_TYPE;
    COLOR_BUFFER : array (0..24, 0..79) of SCREEN_CHAR_TYPE;

    for MONO_BUFFER use at MONO_BUFFER_ADDR;
    for COLOR_BUFFER use at COLOR_BUFFER_ADDR;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure INIT_VIDEO is
        REGISTER  : INTERRUPT.REGISTERS;
        TEST_BITS  : integer;
        EQUIP : EQUIPMENT.EQUIPMENT_LIST;
    begin
        REGISTER.AX := 16#0F00#; -- Return current video state
        INTERRUPT.VECTOR(16#10#, REGISTER);
        if (REGISTER.AX and 16#FF#) = 7 then
            MONOCHROME := true;
        else
            MONOCHROME := false;
        end if;

        REGISTER.AX := 16#1200#; -- Code to return EGA info
        REGISTER.BX := 16#7F10#;
        REGISTER.CX := 16#00FF#;
        INTERRUPT.VECTOR(16#10#, REGISTER);
        TEST_BITS := (REGISTER.BX and 16#7F00#) or (REGISTER.CX and 16#00F0#);
        if TEST_BITS /= 0 then
            EGA_FLAG := false;
        else
            EGA_FLAG := true;
        end if;

        EQUIP := EQUIPMENT.LIST;
        if EQUIP.MATH_COPROCESSOR then
            COPROCESSOR_FLAG := true;
        else
            COPROCESSOR_FLAG := false;
            text_io.put_line("ERROR: No 80x87 math coprocessor found.");
            text_io.put_line("This program requires a math coprocessor.");
            PROGRAM_CONTROL.QUIT(0);
        end if;
    end INIT_VIDEO;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure DRAW_BOX is
    begin
        BOX.DRAW(0, 0, 24, 79, BOX.DOUBLE_SIDED);
    end DRAW_BOX;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function IS_MONOCHROME
        return boolean is
    begin
        return MONOCHROME;
    end IS_MONOCHROME;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure CLEAR_SCREEN is
        REGISTER   : INTERRUPT.REGISTERS;
    begin
        REGISTER.AX := 16#0600#;

        if IS_MONOCHROME then
            REGISTER.BX := 16#0700#;
        else
            REGISTER.BX := 16#0700#;
        end if;

        REGISTER.CX := 16#0000#;
        REGISTER.DX := 16#100# * 24 + 79;
        INTERRUPT.VECTOR(16#10#, REGISTER);
    end CLEAR_SCREEN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure REVERSE_VIDEO_ON is
    begin
        REVERSE_VIDEO := true;
    end REVERSE_VIDEO_ON;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure REVERSE_VIDEO_OFF is
    begin
        REVERSE_VIDEO := false;
    end REVERSE_VIDEO_OFF;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure MOVE_CURSOR (
        ROW     : integer;
        COLUMN   : integer) is

        REGISTER   : INTERRUPT.REGISTERS;
    begin
        REGISTER.AX := 16#0200#;
        REGISTER.BX := 16#0000#;
        REGISTER.DX := 16#100# * ROW + COLUMN;

        INTERRUPT.VECTOR(16#10#, REGISTER);
    end MOVE_CURSOR;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure TURN_CURSOR_ON is
        REGISTER  : INTERRUPT.REGISTERS;
    begin
        REGISTER.AX := 16#0100#; -- Set cursor type
        if IS_MONOCHROME then
            REGISTER.CX := 16#0C0D#; -- Cursor start & stop lines
        else
            REGISTER.CX := 16#0607#;
        end if;
        INTERRUPT.VECTOR(16#10#, REGISTER);
    end TURN_CURSOR_ON;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure TURN_CURSOR_OFF is
        REGISTER  : INTERRUPT.REGISTERS;
    begin
        REGISTER.AX := 16#0100#; -- Set cursor type
        REGISTER.CX := 16#2020#; -- Cursor start & stop lines
        INTERRUPT.VECTOR(16#10#, REGISTER);
    end TURN_CURSOR_OFF;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure PUT_STRING (
        SCREEN_ROW : integer;
        SCREEN_COLUMN : integer;
        TEXT_STRING : string) is

        COLUMN  : integer;
    begin
        if FAST_VIDEO then
            COLUMN := SCREEN_COLUMN;
            if IS_MONOCHROME then
                for I in TEXT_STRING'range loop
                    MONO_BUFFER(SCREEN_ROW, COLUMN).CHAR :=
                    byte(character'pos(TEXT_STRING(I)));
                    if REVERSE_VIDEO then
                        MONO_BUFFER(SCREEN_ROW, COLUMN).ATTRIBUTE := 16#70#;
                    else
                        MONO_BUFFER(SCREEN_ROW, COLUMN).ATTRIBUTE := 16#07#;
                    end if;
                    COLUMN := COLUMN + 1;
                end loop;
            else
                for I in TEXT_STRING'range loop
                    COLOR_BUFFER(SCREEN_ROW, COLUMN).CHAR :=
                    byte(character'pos(TEXT_STRING(I)));
                    if REVERSE_VIDEO then
                        COLOR_BUFFER(SCREEN_ROW, COLUMN).ATTRIBUTE := 16#17#;
                    else
                        COLOR_BUFFER(SCREEN_ROW, COLUMN).ATTRIBUTE := 16#07#;
                    end if;
                    COLUMN := COLUMN + 1;
                end loop;
            end if;
        else
            TTY.Put(COMMON_DISPLAY_TYPES.ROW_RANGE(SCREEN_ROW),
                COMMON_DISPLAY_TYPES.COLUMN_RANGE(SCREEN_COLUMN),
                TEXT_STRING, REVERSE_VIDEO => REVERSE_VIDEO);
        end if;
    end PUT_STRING;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure PUT_REAL (
        ROW  : integer;
        COLUMN  : integer;
        NUMBER  : REAL;
        AFTER  : integer := 2) is

        REAL_STRING : string(1..10);
    begin
        REAL_IO.Put(REAL_STRING, NUMBER, AFT => AFTER, EXP => 0);
        PUT_STRING(ROW, COLUMN, REAL_STRING);
    end PUT_REAL;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure INPUT_STRING (
        TEXT_STRING : out string) is
        LAST : natural;
    begin
        LAST := TEXT_STRING'last;
        TTY.Get(TEXT_STRING, LAST);
    end INPUT_STRING;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function GET_KEY
        return character is
        SCAN_CODE : COMMON_DISPLAY_TYPES.byte;
        CHAR : character;
    begin
        TTY.Get(SCAN_CODE, CHAR);
        return CHAR;
    end GET_KEY;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function KEY_AVAILABLE
        return boolean is
    begin
        return TTY.CHAR_READY;
    end KEY_AVAILABLE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function GET_MENU_COMMAND
        return MENU_COMMAND is

        CHAR : character;
        SCAN_CODE : COMMON_DISPLAY_TYPES.byte;
        NEW_MENU_COMMAND : MENU_COMMAND;
    begin
        loop
            TTY.Get(SCAN_CODE, CHAR);

            case SCAN_CODE is
                when 72 =>
                    NEW_MENU_COMMAND := UP_ARROW;
                     exit;
                when 80 =>
                    NEW_MENU_COMMAND := DOWN_ARROW;
                     exit;
                when 28 =>
                    NEW_MENU_COMMAND := ENTER_KEY;
                     exit;
                when others =>
                    null;
            end case;
        end loop;
        return NEW_MENU_COMMAND;
    end GET_MENU_COMMAND;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure QUIT_PROGRAM is
        DUMMY : integer;
    begin
```

```
        CLEAR_SCREEN;
        MOVE_CURSOR(0, 0);
        TURN_CURSOR_ON;
        PROGRAM_CONTROL.QUIT(0);
    end QUIT_PROGRAM;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
end SYSTEM_SPECIFIC;
-------------------------------------------------------------------------------
-- Integration Package Specification
--
-- This package contains the limited private data type for state variables and
-- the integration initialization and update procedures used on the states.
--
-------------------------------------------------------------------------------

with MATH; use MATH;

package INTEGRATION is

    procedure INITIALIZE;

    procedure ADVANCE_TIME;

    procedure SET_TIME_STEP_SIZE (
        NEW_TIME_STEP_SIZE : in REAL);

    function STEP_SIZE
    return REAL;

end INTEGRATION;
-------------------------------------------------------------------------------
-- Integration Package Body
--
-- This package performs integration of the state variables.
--
-------------------------------------------------------------------------------

with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;
with APPLICATION;
with ENVIRONMENT;

package body INTEGRATION is
    TIME_STEP_SIZE : REAL;

    STATE : VECTOR(1..APPLICATION.NUMBER_OF_STATE_VARIABLES);
    TEMP_STATE : VECTOR(1..APPLICATION.NUMBER_OF_STATE_VARIABLES);
    DERIVATIVE1 : VECTOR(1..APPLICATION.NUMBER_OF_STATE_VARIABLES);
    DERIVATIVE2 : VECTOR(1..APPLICATION.NUMBER_OF_STATE_VARIABLES);
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure SET_TIME_STEP_SIZE (
        NEW_TIME_STEP_SIZE : in REAL) is
    begin
        TIME_STEP_SIZE := NEW_TIME_STEP_SIZE;
    end SET_TIME_STEP_SIZE;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure HANNA is
    begin
        TEMP_STATE := STATE + DERIVATIVE2 * TIME_STEP_SIZE; -- Euler step
        ENVIRONMENT.SET_TIME(ENVIRONMENT.Time + TIME_STEP_SIZE); -- Update time
        APPLICATION.PUT_STATES(TEMP_STATE);
        APPLICATION.COMPUTE_DERIVATIVES;
        APPLICATION.GET_DERIVATIVES(DERIVATIVE1);
        -- Trapezoidal step
        STATE := STATE + (DERIVATIVE1 + DERIVATIVE2) * 0.5*TIME_STEP_SIZE;
        DERIVATIVE2 := DERIVATIVE1; -- Save derivatives for next step
    end HANNA;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure INITIALIZE is
    begin
        APPLICATION.GET_DERIVATIVES(DERIVATIVE2);
        APPLICATION.GET_STATES(STATE);
    end INITIALIZE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure ADVANCE_TIME is
    begin
        HANNA;
    end ADVANCE_TIME;

    function STEP_SIZE
        return REAL is
    begin
        return TIME_STEP_SIZE;
    end STEP_SIZE;

end INTEGRATION;
--------------------------------------------------------------------------------
-- Math Package Specification
--
--------------------------------------------------------------------------------

with MATH_LIB;

package MATH is
    type REAL is digits 15;   -- Use this for all floating point values

    PI  : constant := MATH_LIB.PI; -- Dimensionless
    E  : constant := MATH_LIB.E; -- Dimensionless
    DEG_TO_RAD : constant := PI / 180.0; -- Deg
    RAD_TO_DEG : constant := 1.0 / DEG_TO_RAD; -- 1 / Deg
    G  : constant := 32.174;  -- Gravity, feet / sec**2
    C  : constant := 983_569_000.0; -- Light speed, feet / sec
    GAMMA : constant := 1.4;  -- Air specific heat ratio
    R  : constant := 1718.0;  -- Air gas constant, foot-lb / (slug-R)
    R_EARTH : constant := 20_925_626.0; -- Earth radius, feet
    BOLTZMANN_K : constant := 1.38E-23;  -- Boltzmann's constant, watt-sec/deg K
    FEET_PER_NMI: constant := 6076.115;  -- Feet / NMI
    MENU_SYSTEM : constant boolean := true;

    function SIGN (
        ARG : in REAL)
    return REAL;

    function SIN (
        ARG : in REAL)
    return REAL;

    function COS (
        ARG : in REAL)
    return REAL;

    function TAN (
        ARG : in REAL)
    return REAL;

    function ASIN (
        ARG : in REAL)
    return REAL;

    function ACOS (
        ARG : in REAL)
    return REAL;
    function ATAN (
        ARG : in REAL)
    return REAL;

    function ATAN2 (
        LEFT : in REAL;
        RIGHT : in REAL)
    return REAL;

    function EXP (
        ARG : in REAL)
    return REAL;

    function SQRT (
        ARG : in REAL)
    return REAL;

    function LN (
        ARG : in REAL)
    return REAL;

    function LOG (
        ARG : in REAL)
    return REAL;

    procedure SEED_RAN (
        ARG : in REAL);

    function RAN
    return REAL;

    function GAUSSIAN (
        MEAN  : in REAL;
        STD_DEVIATION : in REAL)
    return REAL;

    function LIMIT (
        VARIABLE : in REAL;
        LOWER_LIMIT : in REAL;
        UPPER_LIMIT : in REAL)
    return REAL;

    function LIMIT (
        VARIABLE : in REAL;
        LIMIT_MAGNITUDE : in REAL)
    return REAL;

    function "**" (
        LEFT : in REAL;
        RIGHT : in REAL)
    return REAL;

    function MIN (
        LEFT : in REAL;
        RIGHT : in REAL)
    return REAL;

    function MAX (
        LEFT : in REAL;
        RIGHT : in REAL)
    return REAL;

    function MAX3 (
        LEFT : in REAL;
        MID : in REAL;
        RIGHT : in REAL)
    return REAL;
end MATH;
```

```
-- Below are instantiations of generic packages

with MATH; use MATH;
with Text_io; use Text_io;

package REAL_IO is new Float_io(REAL);

with Text_io;
package LONG_IO is new Text_io.Integer_io(long_integer);

-- Define a package of 'glue' routines needed for REAL_MATRIX

with Text_io; use Text_io;
with REAL_IO; use REAL_IO;
with MATH; use MATH;

package REAL_MATRIX_GLUE is
    function MAGNITUDE(ELEMENT : REAL) return REAL;
    function TO_FLOAT(ELEMENT : REAL) return float;
    function FROM_FLOAT(ELEMENT : float) return REAL;

    procedure GET_ELEMENT(FILE : in File_type; ELEMENT : out REAL);
    procedure GET_ELEMENT(ELEMENT : out REAL);
    procedure PUT_ELEMENT(FILE : in File_type; ELEMENT : in REAL);
    procedure PUT_ELEMENT(ELEMENT : in REAL);

    pragma INLINE (MAGNITUDE, TO_FLOAT, FROM_FLOAT);
end REAL_MATRIX_GLUE;

package body REAL_MATRIX_GLUE is
    function MAGNITUDE (ELEMENT : REAL) return REAL is
    begin
        return abs(ELEMENT);
    end MAGNITUDE;

    function TO_FLOAT (ELEMENT : REAL) return float is
    begin
        return float(ELEMENT);
    end TO_FLOAT;

    function FROM_FLOAT (ELEMENT : float) return REAL is
    begin
        return REAL(ELEMENT);
    end FROM_FLOAT;

    procedure GET_ELEMENT(FILE : in File_type; ELEMENT : out REAL) is
    begin
        Get(FILE, ELEMENT);
    end GET_ELEMENT;

    procedure GET_ELEMENT(ELEMENT : out REAL) is
    begin
        Get(ELEMENT);
    end GET_ELEMENT;

    procedure PUT_ELEMENT(FILE : in File_type; ELEMENT : in REAL) is
    begin
        Put(FILE, ELEMENT);
        New_line(FILE);
    end PUT_ELEMENT;

    procedure PUT_ELEMENT(ELEMENT : in REAL) is
    begin
        Put(ELEMENT);
        New_line;
    end PUT_ELEMENT;
end REAL_MATRIX_GLUE;

-- Instantiate the MATRIX_AND_VECTOR package for type REAL using
-- the REAL_MATRIX_GLUE routines

with MATRIX_AND_VECTOR;
with MATH; use MATH;
with REAL_MATRIX_GLUE; use REAL_MATRIX_GLUE;
package REAL_MATRIX is new MATRIX_AND_VECTOR(REAL);
--------------------------------------------------------------------------------
-- Math Package Body
--
--------------------------------------------------------------------------------

with MATH_LIB; use MATH_LIB;

package body MATH is
    RANDOM_NUMBER : REAL := 0.5; -- Default seed value
    EXTRA_GAUSS  : boolean := false;
    GAUSS_1  : REAL;
    GAUSS_2  : REAL;
    GAUSS_MAG  : REAL;
    GAUSS_FACTOR : REAL;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function SIGN (
        ARG : in REAL)
        return REAL is
    begin
        if ARG > 0.0 then
            return  1.0;
        elsif ARG < 0.0 then
            return -1.0;
        else
            return  0.0;
        end if;
    end SIGN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function SIN (
        ARG : in REAL)
        return REAL is
    begin
        return REAL(SIN(float(ARG)));
    end SIN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function COS (
        ARG : in REAL)
        return REAL is
    begin
        return REAL(COS(float(ARG)));
    end COS;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function TAN (
        ARG : in REAL)
        return REAL is
    begin
        return SIN(ARG) / COS(ARG);
    end TAN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function ASIN (
        ARG : in REAL)
        return REAL is
    begin
        if ARG = 1.0 then
            return 0.5*PI;
        else
            return REAL(ATAN(float(ARG / SQRT(1.0 - ARG*ARG))));
        end if;
    end ASIN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function ACOS (
        ARG : in REAL)
        return REAL is
    begin
        if ARG = 0.0 then
            return 0.5*PI;
        else
            return REAL(ATAN(float(SQRT(1.0 - ARG*ARG) / ARG)));
        end if;
    end ACOS;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function ATAN (
        ARG : in REAL)
        return REAL is
    begin
        return REAL(ATAN(float(ARG)));
    end ATAN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function ATAN2 (
        LEFT : in REAL;
        RIGHT : in REAL)
        return REAL is

        RESULT : REAL;
    begin
        if RIGHT /= 0.0 then
            RESULT := ATAN(LEFT / RIGHT);
        elsif LEFT > 0.0 then
            return  0.5*PI;
        else
            return -0.5*PI;
        end if;

        if RIGHT < 0.0 then
            if LEFT > 0.0 then
                RESULT := RESULT + PI;
            else
                RESULT := RESULT - PI;
            end if;
        end if;

        return RESULT;
    end ATAN2;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function EXP (
        ARG : in REAL)
        return REAL is
    begin
        return REAL(EXP(float(ARG)));
    end EXP;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function SQRT (
        ARG : in REAL)
        return REAL is
    begin
        if ARG < 0.0 then
            raise Constraint_error;
        end if;
        return REAL(SQRT(float(ARG)));
    end SQRT;
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
    function LN (
        ARG : in REAL)
        return REAL is
    begin
        return REAL(LN(float(ARG)));
    end LN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function LOG (
        ARG : in REAL)
        return REAL is
    begin
        return LN(ARG) / LN(10.0);
    end LOG;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SEED_RAN (
        ARG : in REAL) is
    begin
        RANDOM_NUMBER := abs(ARG);
    end SEED_RAN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function RAN
        return REAL is
        ------------------------------------------------------------------------
        -- Notes on this function: Using the default seed (0.5) -
        --
        -- Using a multiplier of 257.874562956093 a pattern of dots created
        -- on an EGA (640x350) screen by randomly generating x and y point
        -- coordinates had visible line patterns.
        --
        -- Using a multiplier of 19257.874562956093 the random number sequence
        -- began repeating after about 2 million calls.
        --
        -- Using the current value (1257.874562956093) the sequence will
        -- go at least 4 million calls without repeating & appears random on
        -- the EGA.
        ------------------------------------------------------------------------
    begin
        RANDOM_NUMBER := (RANDOM_NUMBER + 0.18568271032) * 1257.874562956093;
        RANDOM_NUMBER := RANDOM_NUMBER - REAL(integer(RANDOM_NUMBER-0.5));
        return RANDOM_NUMBER;
    end RAN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function GAUSSIAN (
        MEAN  : in REAL;
        STD_DEVIATION : in REAL)
        return REAL is
    begin
        if not EXTRA_GAUSS then
            loop
                GAUSS_1 := 2.0 * RAN - 1.0;
                GAUSS_2 := 2.0 * RAN - 1.0;

                GAUSS_MAG := GAUSS_1**2 + GAUSS_2**2;
                exit when GAUSS_MAG < 1.0;
            end loop;

            GAUSS_FACTOR := SQRT(-2.0 * LN(GAUSS_MAG) / GAUSS_MAG);
            GAUSS_1 := MEAN + STD_DEVIATION * GAUSS_1 * GAUSS_FACTOR;

            EXTRA_GAUSS := true;
            return GAUSS_1;
        else
            GAUSS_2 := MEAN + STD_DEVIATION * GAUSS_2 * GAUSS_FACTOR;
            EXTRA_GAUSS := false;
            return GAUSS_2;
        end if;
    end GAUSSIAN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function LIMIT (
        VARIABLE : in REAL;
        LOWER_LIMIT : in REAL;
        UPPER_LIMIT : in REAL)
        return REAL is
    begin
        if VARIABLE > UPPER_LIMIT then
            return UPPER_LIMIT;
        elsif VARIABLE < LOWER_LIMIT then
            return LOWER_LIMIT;
        else
            return VARIABLE;
        end if;
    end LIMIT;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function LIMIT (
        VARIABLE : in REAL;
        LIMIT_MAGNITUDE : in REAL)
        return REAL is
    begin
        if VARIABLE > LIMIT_MAGNITUDE then
            return  LIMIT_MAGNITUDE;
        elsif VARIABLE < -LIMIT_MAGNITUDE then
            return -LIMIT_MAGNITUDE;
        else
            return  VARIABLE;
        end if;
    end LIMIT;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function "**" (
        LEFT : in REAL;
        RIGHT : in REAL)
        return REAL is

        SIGN : REAL;
    begin
        if LEFT = 0.0 then
            return 0.0;
        elsif RIGHT = REAL(integer(RIGHT)) and LEFT < 0.0 then
            if RIGHT = 2.0 * REAL(integer(RIGHT / 2.0)) then -- Even power
                SIGN :=  1.0;
            else  -- Odd power
                SIGN := -1.0;
            end if;

            return SIGN * EXP(RIGHT * LN(abs(LEFT)));
        else
            return EXP(RIGHT * LN(LEFT));
        end if;
    end "**";
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function MIN (
        LEFT : in REAL;
        RIGHT : in REAL)
        return REAL is
    begin
        if LEFT < RIGHT then
            return LEFT;
        else
            return RIGHT;
        end if;
    end MIN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function MAX (
        LEFT : in REAL;
        RIGHT : in REAL)
        return REAL is
    begin
        if LEFT > RIGHT then
            return LEFT;
        else
            return RIGHT;
        end if;
    end MAX;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function MAX3 (
        LEFT : in REAL;
        MID : in REAL;
        RIGHT : in REAL)
        return REAL is
    begin
        return MAX(MAX(LEFT, MID), RIGHT);
    end MAX3;
end MATH;
--------------------------------------------------------------------------------
-- Model Data Types Package Specification
--
-- This package contains the data types for application specific variables
-- that need to be defined in several places in the simulation
--
--------------------------------------------------------------------------------

with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;

package MODEL_TYPES is
    type YES_NO_TYPE is (NO, YES);
    type SUPER_VECTOR is array (positive range <>) of VECTOR(1..3);
    type THREED is array (positive range <>,positive range <>, positive
                                  range <>) of REAL;

    type IR_PHASE_TYPE is (NON_ACTIVE, IR_SEARCH, IR_ACQUISITION, IR_TRACK);

    type RF_PHASE_TYPE1 is (MIDCOURSE,HPRF_ACQUISTION,MPRF_ACQUISTION,TERMINAL);
    type RF_PHASE_TYPE2 is (NON_ACTIVE,MIDCOURSE,X_BAND_ACQUISITION,
        X_BAND_TRACK,K_BAND_ACQUISITION,K_BAND_TRACK);

    type AIRCRAFT_TYPE is (F_14,F_15,F_18,SIXDOF);
    type LAUNCHER_TYPE is (RAIL, EJECT);

    type SSJ_ECM_TYPE is (NONE,REPEATER,BARRAGE_NOISE);
    type SOJ_ECM_TYPE is (BARRAGE_NOISE,RSN,PRN);
    type SSJ_ECM_TECHNIQUE is array (1..2) of SSJ_ECM_TYPE;
    type SOJ_ECM_TECHNIQUE is array (1..2) of SOJ_ECM_TYPE;

    type GUIDANCE_PHASE_TYPE is (NULL_COMMANDS, HOLD_PATH, LOAD_BIAS,
        VARIABLE_ARC, ALTITUDE_HOLD, TURN_DOWN, TERMINAL);

    type GUID_SELECTION_TYPE is (LAC_CUE,TGT_1,TGT_2,SOJ_1,SOJ_2);

    type PROPUL_TYPE is (BOOST,COAST,FLAMEOUT);

    type INT_GUIDANCE_TYPE is (NON_MANEUVERING, PURSUIT);

    type TGT_IR_SIZE_TYPE is (SMALL, MEDIUM, LARGE);
    type MANEUVER_TYPE is (NONE,TURN,WEAVE);
    type MANEUVER_START_TYPE is (FLIGHT_TIME,TIME_REMAINING,RANGE_TO_GO);
type STOP_CONDITION_TYPE is (AZ_GIMBAL,EL_GIMBAL,TOTAL_GIMBAL,RANGE_RATE,
    CROSSOVER,HIT_GROUND,MAX_TIME_OF_FLIGHT);

type SETUP_VALUES_TYPE is
record
     FRAME_TIME : REAL;
     LOG_INTERVAL : INTEGER;
    INT_ALTITUDE_IC : REAL;
    INT_MACH_IC : REAL;
    INT_LEAD_ANGLE_IC : REAL;
     LAUNCH_TYPE      : LAUNCHER_TYPE;
    TGT_ALTITUDE_IC : VECTOR(1..4);
     SOJ_ANGLE_IC     : VECTOR(1..2);
    TGT_MACH_IC : REAL;
    TGT_ASPECT_IC : REAL;
     TGT2_ANGLE_IC : REAL;
    TGT_RANGE_IC : VECTOR(1..4);
    TGT_RCS_IC : vector(1..2);
    TGT_TURN_G_IC : REAL;
    TGT_TURNON_VALUE_IC : REAL;
    TGT_BUILDUP_TIME_IC : REAL;
    TGT_TURN_ANG_IC : REAL;
    TGT_WEAVE_PER_IC : REAL;
     TGT_TWO_IC       : YES_NO_TYPE;
     SOJ_ONE_IC       : YES_NO_TYPE;
     SOJ_TWO_IC       : YES_NO_TYPE;
     TGT_ECM_POWER_IC:  VECTOR(1..4);
     TGT_ECM_TECH_IC:  SSJ_ECM_TECHNIQUE;
     SOJ_ECM_TECH_IC:  SOJ_ECM_TECHNIQUE;
     INT_TYPE_IC  :   AIRCRAFT_TYPE;
    LOG_DATA : YES_NO_TYPE;
    INT_GUIDANCE : INT_GUIDANCE_TYPE;
    TGT1_IR_SIZE : TGT_IR_SIZE_TYPE;
    TGT2_IR_SIZE : TGT_IR_SIZE_TYPE;
    MANEUVER_KIND : MANEUVER_TYPE;
    TURN_ON_PARAMETER : MANEUVER_START_TYPE;
    OUTPUT_FILE : string(1..30);
end record;

type MSL_LOG_DATA_TYPE is
record
    REAL_VALUE : VECTOR(1..34);

    GUIDANCE_PHASE : GUIDANCE_PHASE_TYPE;
    RF_PHASE_1 : RF_PHASE_TYPE1;
    RF_PHASE_2 : RF_PHASE_TYPE2;
    IR_PHASE : IR_PHASE_TYPE;
    RADOME_OFF : boolean;
    PROPULSION_PHASE : PROPUL_TYPE;
end record;

pragma PACK(MSL_LOG_DATA_TYPE);

type LAUNCHER_LOG_DATA_TYPE is
record
    REAL_VALUE : VECTOR(1..3);
end record;

pragma PACK(LAUNCHER_LOG_DATA_TYPE);

type TARGET_LOG_DATA_TYPE is
record
    REAL_VALUE : VECTOR(1..9);
end record;

pragma PACK(TARGET_LOG_DATA_TYPE);

type LOG_RECORD_TYPE is
record
```

```
        MISSILE_DATA : MSL_LOG_DATA_TYPE;
        LAUNCHER_DATA : LAUNCHER_LOG_DATA_TYPE;
        TARGET_DATA : TARGET_LOG_DATA_TYPE;
    end record;

    pragma PACK(LOG_RECORD_TYPE);

    RF_MODE  : array (RF_PHASE_TYPE2) of string(1..9) := (
        "Inactive ",
        "Midcourse",
        "XBand Acq",
        "XBand Trk",
        "KBand Acq",
        "KBand Trk");

    IR_MODE  : array (IR_PHASE_TYPE) of string(1..9) := (
        "Inactive ",
        "IR Search",
        "IR Acq   ",
        "IR Track ");

    PROPULSION_MODE : array (PROPUL_TYPE) of string(1..9) := (
        "Boost    ",
        "Coast    ");

    GUIDANCE_MODE : array (GUIDANCE_PHASE_TYPE) of string(1..9) := (
        "Null Cmds",
        "Hold Path",
        "Load Bias",
        "Var. Arc ",
        "Alt. Hold",
        "Turn Down",
        "Terminal ");

    STOP_REASON : array (STOP_CONDITION_TYPE) of string(1..9) := (
        "Az Gimbal",
        "El Gimbal",
        "Tot Gimb.",
        "Rangerate",
        "Crossover",
        "Hit Grnd.",
        "Max TOF  ");

end MODEL_TYPES;
```

APPENDIX D

PROBLEM SPACE OBJECT SOURCE CODE LISTINGS

```
-------------------------------------------------------------------------
-- Launcher package
--
-- This package contains the procedure calls which will initialize,
-- setup, computes and get and put the missile state vector as well
-- as calculating the necessary parameters to calculate the state
-- vector.
-------------------------------------------------------------------------

with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;
with MODEL_TYPES; use MODEL_TYPES;

package LAUNCHER is

    procedure SETUP(LAC_GUID_IN:  in INT_GUIDANCE_TYPE;
                LEAD_IN,
                LAC_MACH_IN,
                LAC_ALT_IN:in REAL;
                LAC_TYPE_IN:  in AIRCRAFT_TYPE);

    procedure PUT_STATES(STATES:  in VECTOR);

    function GET_DERIVATIVES return VECTOR;

    function GET_STATES return VECTOR;

    function LOG_DATA return LAUNCHER_LOG_DATA_TYPE;

    function POLE(I:  in INTEGER) return REAL;

    procedure INITIALIZE;

    function LAC_POS return VECTOR;

    procedure MSL_INIT(LAC_VEL,LAC_POSITION:  out VECTOR; LAC_GAIN,
                           LAC_TRANS_PWR, LAC_BEAMWIDTH:  out REAL);

    procedure COMPUTE;

end LAUNCHER;

--------------------------------------------------------------------------------
-- Launcher package
--
-- This package contains the procedure calls which will initialize,
-- setup, computes and get and put the missile state vector as well
-- as computing the necessary parameters to calculate the state
-- vector.
--
--------------------------------------------------------------------------------

with MODEL_TYPES; use MODEL_TYPES;
with MATH; use MATH;
with ENVIRONMENT;
with REAL_MATRIX; use REAL_MATRIX;
with MISSILE;
with TARGETS;
WITH TEXT_IO;WITH REAL_IO;

package body LAUNCHER is
```

```
    LAC_GUID        :   INT_GUIDANCE_TYPE; -- Launch Aircraft Guidance Mode
    LEAD            :   REAL;            -- Launch Aircraft Lead Angle
    LAC_MACH        :   REAL;            -- Launch Aircraft Mach
    LAC_VEL_NED     :   VECTOR (1..3);  -- Launch Aircraft Velocity Vector
    LAC_POS_NED     :   VECTOR (1..3);  -- Launch Aircraft Position Vector
    LOGGED_DATA     :   LAUNCHER_LOG_DATA_TYPE ;  -- Output Vector
    DERIVATIVES     :   VECTOR(1..3);  --
    STATE           :   VECTOR(1..3);  --
    LAC_TYPE         :   AIRCRAFT_TYPE;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
    procedure SETUP(LAC_GUID_IN:  in INT_GUIDANCE_TYPE;
                LEAD_IN,
                LAC_MACH_IN,
               LAC_ALT_IN: in REAL;
                LAC_TYPE_IN:  in AIRCRAFT_TYPE) is

    begin
        LAC_GUID  :=  LAC_GUID_IN;   -- Launch Aircraft Guidance Mode
        LEAD      :=  LEAD_IN;   -- Launch Aircraft Lead Angle (rad)
        LAC_MACH  :=  LAC_MACH_IN;   -- Launch Aircraft Mach
         LAC_POS_NED(3):= -LAC_ALT_IN;-- Launch aircraft altitude (feet)
         LAC_TYPE:=LAC_TYPE_IN;
    end SETUP;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
    procedure PUT_STATES(STATES: in VECTOR) is
    begin
        LAC_POS_NED:=STATES(1..3);
    end PUT_STATES;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
    function GET_DERIVATIVES return VECTOR is
    begin
        DERIVATIVES(1..3):=LAC_VEL_NED;
        return DERIVATIVES;
    end GET_DERIVATIVES;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
    function GET_STATES return VECTOR is
    begin
        STATE(1..3):=LAC_POS_NED;
        return STATE;
    end GET_STATES;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
    function LOG_DATA return LAUNCHER_LOG_DATA_TYPE is
    begin
        LOGGED_DATA.REAL_VALUE(1):=  LAC_POS_NED(1);
        LOGGED_DATA.REAL_VALUE(2):=  LAC_POS_NED(2);
        LOGGED_DATA.REAL_VALUE(3):= -LAC_POS_NED(3);
        return LOGGED_DATA;
    end LOG_DATA;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
    function POLE(I: in INTEGER) return REAL is
    TGT_POS_NED : SUPER_VECTOR(1..3);
    begin
         TGT_POS_NED:=TARGETS.TGT_POS;
        return MAGNITUDE(TGT_POS_NED(I) - LAC_POS_NED);
    end POLE;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
    function LAC_POS return VECTOR is
    begin
         return LAC_POS_NED;
    end LAC_POS;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
```

```
    procedure INITIALIZE is
        VSS:  REAL;
    begin
        VSS:=ENVIRONMENT.SPEED_OF_SOUND(-LAC_POS_NED(3));
        LAC_VEL_NED(1):=LAC_MACH*VSS*COS(LEAD);
        LAC_VEL_NED(2):=LAC_MACH*VSS*SIN(LEAD);
        LAC_VEL_NED(3):= 0.0;
        LAC_POS_NED(1):= 0.0;
        LAC_POS_NED(2):= 0.0;
    end INITIALIZE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure MSL_INIT(LAC_VEL,LAC_POSITION : out VECTOR; LAC_GAIN,
                LAC_TRANS_PWR, LAC_BEAMWIDTH:  out REAL) is
    begin
        LAC_VEL:=LAC_VEL_NED;
        LAC_POSITION:=LAC_POS_NED;

        case LAC_TYPE is
            when F_14 =>
                LAC_TRANS_PWR := 32.0;     -- dbw
                LAC_GAIN      := 36.5;     -- db
                LAC_BEAMWIDTH := 2.2;      -- deg
            when F_15 =>
                LAC_TRANS_PWR := 26.0;     -- dbw
                LAC_GAIN      := 36.0;     -- db
                LAC_BEAMWIDTH := 2.3;      -- deg
            when F_18 =>
                LAC_TRANS_PWR := 25.0;     -- dbw
                LAC_GAIN      := 33.0;     -- db
                LAC_BEAMWIDTH := 3.4;      -- deg
            when others =>
        -- use for 6dof comparison
                LAC_TRANS_PWR := 37.0;     -- dbw
                LAC_GAIN      := 37.0;     -- db
        -- not an actual value will replace once found
                LAC_BEAMWIDTH := 3.0;      -- deg
        end case;
    end MSL_INIT;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure COMPUTE is
        DIFF : VECTOR(1..3);
        ANGLE       : REAL;
        LAC_VEL  : REAL;
        TGT_POS_NED : SUPER_VECTOR(1..3);

     begin
        if LAC_GUID = PURSUIT then
           TGT_POS_NED := TARGETS.TGT_POS;
           DIFF:=TGT_POS_NED(1)-LAC_POS_NED;
           ANGLE:=ATAN(DIFF(2)/DIFF(1));
           LAC_VEL:=MAGNITUDE(LAC_VEL_NED);
           LAC_VEL_NED(1):=LAC_VEL*COS(ANGLE);
           LAC_VEL_NED(2):=LAC_VEL*SIN(ANGLE);
        end if;
     end COMPUTE;

end LAUNCHER;

-----------------------------------------------------------------------
-- Missile package
--
-- This package contains the procedure calls which will initialize,
-- setup, compute and get and put the missile state vector, as well
-- as calculating the necessary parameters to calculate the state
-- vector.
--
-----------------------------------------------------------------------
```

```
with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;
with MODEL_TYPES; use MODEL_TYPES;

package MISSILE is

    ID_STRING : string(1..30) := "NPS Missile Flight Simulation ";
    RUNTIME_TITLE : string(1..42) :=
        "       NPS Simulation Runtime Display      ";
    OUTFILE : string(1..30) := "SIM.OUT                       ";
    DATFILE : string(1..30) := "SIM.DAT                       ";
    MAIN_MENU_TITLE : string(1..60) :=
        "NPS 3-DOF Missile Flight Simulation Main Menu           ";
    TITLE_SCREEN_LINE : string(1..60) :=
        "     NPS 3-DOF Air-to-Air Missile Flight Simulation      ";

    IS_ACTIVE : boolean := true; -- If true this is an active missile

    procedure SETUP(LAUNCH_TYPE_IN:       in LAUNCHER_TYPE;
                    RCS:                  in VECTOR;
                    TGT1_IR_SIGNATURE:  in TGT_IR_SIZE_TYPE;
                    TGT2_IR_SIGNATURE:  in TGT_IR_SIZE_TYPE;
                    TGT_TWO:              in YES_NO_TYPE;
                    SOJ_ONE:              in YES_NO_TYPE;
                    SOJ_TWO:              in YES_NO_TYPE;
                    ECM_POWER:            in VECTOR;
                    SSJ_ECM_TECH:         in SSJ_ECM_TECHNIQUE;
                    SOJ_ECM_TECH:         in SOJ_ECM_TECHNIQUE);

    procedure INITIALIZE;

    procedure COMPUTE;

    procedure PUT_STATES(STATE:  in VECTOR);

    procedure MANEUVER_VALUE(TURN_ON_PARAMETER:  in MANEUVER_START_TYPE;
        MANEUVER_START_VALUE:  out REAL);

    function GET_DERIVATIVES return VECTOR;

    function GET_STATES return VECTOR;

    function LOG_DATA return MSL_LOG_DATA_TYPE;

    function END_CONDITIONS_MET return boolean;

    procedure TERMINAL_CONDITIONS (A_POLE_OUT, MISS_DISTANCE_OUT,
                                   TIME_OF_FLIGHT, ALTITUDE, RDOT: out REAL;
                                   STOP_REASON_OUT:  out STOP_CONDITION_TYPE);

end MISSILE;

--------------------------------------------------------------------------------
-- Missile package body
--
-- This package contains the procedure calls which will initialize,
-- setup, compute and get and put the missile state vector, as well
-- as computing the necessary parameters to calculate the state
-- vector.
--------------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;
with LAUNCHER;
with TARGETS;
with AIRFRAME;
with AUTOPILOT;
with ENVIRONMENT;
with GUIDANCE;
with INTEGRATION;
with KINEMATICS;
with RF_SEEKER;
with IR_SEEKER;

package body MISSILE is
    MSL_ACC_NED    : VECTOR(1..3);   -- Missile acceleration vector
    MSL_VEL_NED    : VECTOR(1..3);   -- Missile velocity vector
    MSL_POS_NED    : VECTOR(1..3);   -- Missile position vector
    LAUNCH_TYPE     : LAUNCHER_TYPE;
    NTGTS           : INTEGER;     -- Number of targets
    NSOJS           : INTEGER;     -- Number of active stand off jammers
    TGTTYP          : INTEGER;     -- Type of target (skin,range deception, etc.)
    BORE_SIGHT_ERROR:VECTOR(1..4);-- Bore sight errors
    MSL_DATA       : MSL_LOG_DATA_TYPE; -- Missile object output record
    DERIV          : VECTOR(1..6); -- Derivative array vector
    STATE_OUT      : VECTOR(1..6); -- State variable output vector
    TGT_VEL_NED    : VECTOR(1..3);   --   Target velocity vectors, ft/sec
    TGT_POS_NED    : SUPER_VECTOR(1..4);   --   Target position vectors, ft
    IR_ACQ_RANGE   :       REAL; -- Range at which IR acquisition is enabled
    ANGLE_OF_ATTACK  : VECTOR(1..3); --Missile body angle vector
    COEF_DRAG        : REAL; -- Missile drag coefficient
    MSL_MASS         : REAL; -- Missile mass
    THRUST           : REAL; -- Missile thrust, lbs
    MSL_VEL          : REAL;  -- Missile velocity, ft/sec
    MSL_HEADING_AZ   : REAL;  -- Missile azimuth heading angle, rad
    PITCH            : REAL;  -- Missile pitch angle, rad
    RANGE_VEC_NED :   SUPER_VECTOR(1..4);   -- Missile to target range vectors
    OMEGA_NED     :   VECTOR(1..3);   -- Missile to target LOS rates
    TGT_RANGE     :   VECTOR(1..4);        -- Missile to target ranges, ft
    RDOT          :   VECTOR(1..2);      -- Missile to target range rates, fps
    TIME_TO_GO    :   REAL;            -- Estimated time to go in flight, sec
    RANGE_VEC_BOD : SUPER_VECTOR(1..4); --Missile to target one range vector,
    GMB_ANG_SKR   :VECTOR(1..3); -- Seeker gimbal angle vector, rad
    MSL_PHASE     :   RF_PHASE_TYPE1;         -- Missile phase of flight
    RF_PHASE      :   RF_PHASE_TYPE2;         -- Phase that RF is in
    IR_PHASE      :   IR_PHASE_TYPE;        -- IR phase
    PROPULSION_PHASE: PROPUL_TYPE;
    RADOME_OFF    :   boolean;                  -- Radome off flag
    F_POLE        :   REAL;
    A_POLE        :   REAL;
    RANGE_TGT_LAC :   VECTOR(1..4); -- Range from the targets to the
    --                                  launch aircraft
    SNR           :   REAL; -- Signal to noise ratio, dB
    MSL_AXIAL_ACC :   REAL; -- Missile axial acceleration in body axis
    OMEGA_SKR     :   VECTOR(1..3); -- Missile to targets LOS rate vectors
    GUID_PHASE    :   GUIDANCE_PHASE_TYPE; -- Guidance phase
    ACC_COM_BOD   :   VECTOR(2..3);   -- Commanded acceleration vector
    ACC_ACH_BOD   :   VECTOR(1..3);   -- Achieved acceleration vector- body
    Q             :   REAL;           -- Dynamic pressure
    MSL_MACH      :   REAL;
    STOP_REASON   :   STOP_CONDITION_TYPE; -- Reason simulation stopped
    MISS_DISTANCE_NED:   VECTOR(1..3);
    MISS_DISTANCE :      REAL;
    RDOT_OLD       :     REAL;   -- Range rate value from prior pass
    TIME_TO_GO_OLD:      REAL;
    ALTITUDE_OLD  :      REAL;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SETUP(LAUNCH_TYPE_IN:       in LAUNCHER_TYPE;
                    RCS:                  in VECTOR;
                    TGT1_IR_SIGNATURE:    in TGT_IR_SIZE_TYPE;
                    TGT2_IR_SIGNATURE:    in TGT_IR_SIZE_TYPE;
                    TGT_TWO:              in YES_NO_TYPE;
                    SOJ_ONE:              in YES_NO_TYPE;
                    SOJ_TWO:              in YES_NO_TYPE;
                    ECM_POWER:            in VECTOR;
                    SSJ_ECM_TECH:         in SSJ_ECM_TECHNIQUE;
                    SOJ_ECM_TECH:        in SOJ_ECM_TECHNIQUE) is
    TEMP_ECM_POWER:      VECTOR(1..4);

    begin
        LAUNCH_TYPE:=LAUNCH_TYPE_IN;
        TEMP_ECM_POWER:=ECM_POWER;
        if TGT_TWO = NO then
            NTGTS:=1;
        else
            NTGTS:=2;
        end if;
        if SOJ_ONE = NO and SOJ_TWO = NO then
            NSOJS:=0;
        elsif SOJ_ONE = YES and SOJ_TWO = NO then
            NSOJS:=1;
        elsif SOJ_ONE = NO and SOJ_TWO = YES then
            NSOJS:=1;
            TEMP_ECM_POWER(3):=TEMP_ECM_POWER(4);
        elsif SOJ_ONE = YES and SOJ_TWO = YES then
            NSOJS:=3;
        end if;
        RF_SEEKER.SETUP(NTGTS,NSOJS,RCS,TEMP_ECM_POWER,SSJ_ECM_TECH,
                        SOJ_ECM_TECH);
        IR_SEEKER.SETUP(TGT1_IR_SIGNATURE,TGT2_IR_SIGNATURE);
        KINEMATICS.SETUP(NTGTS,NSOJS);
    end SETUP;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure INITIALIZE is
    GAIN        :       REAL;
    POWER       :       REAL;
    BEAMWIDTH   :       REAL;

    begin
        MSL_MASS:=400.0;
        COEF_DRAG:=0.75;
        THRUST:=0.0;
        ANGLE_OF_ATTACK:=(0.0,0.0,0.0);
        ACC_ACH_BOD := (0.0,0.0,0.0);
        PITCH := 0.0;
        MSL_PHASE:=MIDCOURSE;
        RF_PHASE:=NON_ACTIVE;
        IR_PHASE:=NON_ACTIVE;
        PROPULSION_PHASE:=COAST;
        RADOME_OFF:=false;
        Q:=2600.0;
        IS_ACTIVE:=FALSE;
        for I in 1..4 loop
            RANGE_TGT_LAC(I):=LAUNCHER.POLE(I);
            TGT_RANGE(I):=RANGE_TGT_LAC(I);
        end loop;
        A_POLE:=RANGE_TGT_LAC(1);
        SNR:=1.0;
        TIME_TO_GO:=500.0;
        TARGETS.TGT_DATA(TGT_VEL_NED,TGT_POS_NED);
        LAUNCHER.MSL_INIT(MSL_VEL_NED,MSL_POS_NED,GAIN,POWER,
                                        BEAMWIDTH);
        GUIDANCE.INITIALIZE(-MSL_POS_NED(3),-TGT_POS_NED(1)(3),
                            TGT_POS_NED(1),TGT_VEL_NED);
        AIRFRAME.INITIALIZE;
        AUTOPILOT.INITIALIZE;
        RF_SEEKER.INITIALIZE(GAIN,POWER,BEAMWIDTH);
        IR_SEEKER.INITIALIZE(-TGT_POS_NED(1)(3),-MSL_POS_NED(3),IR_ACQ_RANGE);
        MSL_HEADING_AZ:=ATAN(MSL_VEL_NED(2)/MSL_VEL_NED(1));
        if LAUNCH_TYPE = EJECT then
            MSL_VEL_NED(3):=MSL_VEL_NED(3)+20.0;
        end if;
    end INITIALIZE;
--------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------
procedure COMPUTE is
    EIB       :   MATRIX(1..3,1..3);  -- Inertial to body direction cos mat.
    EIS       :   MATRIX(1..3,1..3);  -- Inertial to seeker dir. cos matrix
    EBI       :   MATRIX(1..3,1..3);  -- Transform of EIB
    THETA_SKR :   REAL;               -- Seeker pitch angle, rad
    PSI_SKR   :   REAL;               -- Seeker yaw angle, rad

begin -- COMPUTE
    TIME_TO_GO_OLD:=TIME_TO_GO;
    RDOT_OLD:=RDOT(1);

    TARGETS.TGT_DATA(TGT_VEL_NED,TGT_POS_NED);
    KINEMATICS.COMPUTE(MSL_POS_NED,MSL_VEL_NED,TGT_VEL_NED,
                       TGT_POS_NED,RANGE_VEC_NED,OMEGA_NED,TGT_RANGE,RDOT,
                       MISS_DISTANCE_NED,TIME_TO_GO,MISS_DISTANCE);

    for I in 1..NTGTS loop
        RANGE_TGT_LAC(I):=LAUNCHER.POLE(I);
    end loop;

    for I in 1.. NSOJS loop
        RANGE_TGT_LAC(I+2):=LAUNCHER.POLE(I+2);
    end loop;

        RF_SEEKER.DETECTION(TGT_RANGE,RANGE_TGT_LAC,RF_PHASE,SNR,
                            TGTTYP,BORE_SIGHT_ERROR);

    case RF_PHASE is
        when K_BAND_ACQUISITION =>
            if not IS_ACTIVE then
                A_POLE:=RANGE_TGT_LAC(1);
                IS_ACTIVE:=true;
            end if;
        when others =>
            null;
    end case;

    if TGT_RANGE(1) < IR_ACQ_RANGE then
        RADOME_OFF:=TRUE;
        IR_SEEKER.DETECTION(TGT_RANGE(1),IR_PHASE);
    end if;

    KINEMATICS.DIR_COS((MSL_HEADING_AZ+ANGLE_OF_ATTACK(3)),PITCH,EIB);

    for I in 1..NTGTS loop
        RANGE_VEC_BOD(I):=EIB*RANGE_VEC_NED(I);
    end loop;

    for I in 1..NSOJS loop
        RANGE_VEC_BOD(I+2):=EIB*RANGE_VEC_NED(I+2);
    end loop;

    RF_SEEKER.GIMBAL(RANGE_VEC_BOD,GMB_ANG_SKR);

    PSI_SKR:=ANGLE_OF_ATTACK(3)+MSL_HEADING_AZ+GMB_ANG_SKR(3);
    THETA_SKR:=PITCH+GMB_ANG_SKR(2);
    KINEMATICS.DIR_COS(PSI_SKR,THETA_SKR,EIS);
    OMEGA_SKR:=EIS*OMEGA_NED;
    GUIDANCE.COMPUTE(TIME_TO_GO,-MSL_POS_NED(3),-MSL_VEL_NED(3),
        MSL_VEL,ACC_ACH_BOD(1),PITCH,TGT_RANGE(1),RDOT(1),
        GMB_ANG_SKR,OMEGA_SKR,GUID_PHASE,ACC_COM_BOD);
    AUTOPILOT.COMPUTE(ACC_COM_BOD,ACC_ACH_BOD(2..3));
    AUTOPILOT.UPDATE_DIFF_EQS;
    KINEMATICS.MACH_NO(-MSL_POS_NED(3),MSL_VEL_NED,MSL_MACH);
    AIRFRAME.THRUST(MSL_MASS,THRUST,PROPULSION_PHASE);
    AIRFRAME.AERO(MSL_MACH,Q,RADOME_OFF,
                  COEF_DRAG,ANGLE_OF_ATTACK);
    KINEMATICS.EOM(MSL_POS_NED,MSL_VEL_NED,ANGLE_OF_ATTACK,
            COEF_DRAG,MSL_MASS,THRUST,ACC_ACH_BOD(1),
            MSL_VEL,MSL_HEADING_AZ,PITCH,Q);
        EBI := TRANSPOSE(EIB);
        MSL_ACC_NED := EBI * ACC_ACH_BOD;
        MSL_ACC_NED(3) := MSL_ACC_NED(3) + G;
    end COMPUTE;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure PUT_STATES(STATE:  in VECTOR) is
    begin
        ALTITUDE_OLD:=-MSL_POS_NED(3);
        MSL_VEL_NED:=STATE(1..3);
        MSL_POS_NED:=STATE(4..6);
    end PUT_STATES;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure MANEUVER_VALUE(TURN_ON_PARAMETER:  in MANEUVER_START_TYPE;
        MANEUVER_START_VALUE:  out REAL) is
    begin
        if TURN_ON_PARAMETER = FLIGHT_TIME then
            MANEUVER_START_VALUE:=ENVIRONMENT.Time;
        elsif TURN_ON_PARAMETER = TIME_REMAINING then
            MANEUVER_START_VALUE:=TIME_TO_GO;
        elsif TURN_ON_PARAMETER = RANGE_TO_GO then
            MANEUVER_START_VALUE:=TGT_RANGE(1) / FEET_PER_NMI;
        end if;
    end MANEUVER_VALUE;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    function GET_DERIVATIVES return VECTOR is
    begin
        DERIV(1..3):=MSL_ACC_NED;
        DERIV(4..6):=MSL_VEL_NED;
        return DERIV;
    end GET_DERIVATIVES;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    function GET_STATES return VECTOR is
    begin
        STATE_OUT(1..3):=MSL_VEL_NED;
        STATE_OUT(4..6):=MSL_POS_NED;
        return STATE_OUT;
    end GET_STATES;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    function LOG_DATA return MSL_LOG_DATA_TYPE is
    begin
        MSL_DATA.REAL_VALUE(1):= MSL_POS_NED(1);
        MSL_DATA.REAL_VALUE(2):= MSL_POS_NED(2);
        MSL_DATA.REAL_VALUE(3):=-MSL_POS_NED(3);
        MSL_DATA.REAL_VALUE(4):=RANGE_TGT_LAC(1);
        MSL_DATA.REAL_VALUE(5):=MSL_VEL;
        MSL_DATA.REAL_VALUE(6):=MSL_HEADING_AZ*RAD_TO_DEG;
        MSL_DATA.REAL_VALUE(7):=PITCH*RAD_TO_DEG;
        MSL_DATA.REAL_VALUE(8):=MSL_MACH;
        MSL_DATA.REAL_VALUE(9):=TGT_RANGE(1);
        MSL_DATA.REAL_VALUE(10):=RDOT(1);
        MSL_DATA.REAL_VALUE(11):=TIME_TO_GO;
        MSL_DATA.REAL_VALUE(12..14):=GMB_ANG_SKR*RAD_TO_DEG;
        MSL_DATA.REAL_VALUE(15..16):=OMEGA_SKR(2..3)*RAD_TO_DEG;
        MSL_DATA.REAL_VALUE(17):=-MSL_VEL_NED(3);
        MSL_DATA.REAL_VALUE(18..19):=ACC_COM_BOD;
        MSL_DATA.REAL_VALUE(20..22):=ACC_ACH_BOD;
        MSL_DATA.REAL_VALUE(23):=MSL_MASS;
        MSL_DATA.REAL_VALUE(24):=THRUST;
        MSL_DATA.REAL_VALUE(25..27):=ANGLE_OF_ATTACK*RAD_TO_DEG;
        MSL_DATA.REAL_VALUE(28):=SNR;
        MSL_DATA.REAL_VALUE(29):=COEF_DRAG;
        MSL_DATA.REAL_VALUE(30):=REAL(TGTTYP);
```

```
         MSL_DATA.REAL_VALUE(31..34):=BORE_SIGHT_ERROR;
         MSL_DATA.GUIDANCE_PHASE := GUID_PHASE;
         MSL_DATA.RF_PHASE_1 := MSL_PHASE;
         MSL_DATA.RF_PHASE_2 := RF_PHASE;
         MSL_DATA.IR_PHASE := IR_PHASE;
         MSL_DATA.RADOME_OFF := RADOME_OFF;
         MSL_DATA.PROPULSION_PHASE := PROPULSION_PHASE;
         return MSL_DATA;
     end LOG_DATA;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
     function END_CONDITIONS_MET return boolean is
     begin
         if abs(GMB_ANG_SKR(2)) >=0.95993 then          -- Azimuth gimbal angle
             STOP_REASON:=EL_GIMBAL;
             return true;
         elsif abs(GMB_ANG_SKR(3)) >= 0.95993 then      -- Elevation gimbal angle
             STOP_REASON:=AZ_GIMBAL;
             return true;
         elsif abs(GMB_ANG_SKR(1)) >= 0.95993 then      -- Total gimbal angle
             STOP_REASON:=TOTAL_GIMBAL;
             return true;
         elsif ENVIRONMENT.TIME > 7.0 and RDOT(1) >= 0.0 then
             if MAGNITUDE(MSL_POS_NED) >= MAGNITUDE(TGT_POS_NED(1)) then
                 STOP_REASON:=CROSSOVER;
             else
                 STOP_REASON:=RANGE_RATE;
             end if;
             return true;
         elsif -MSL_POS_NED(3) <= 0.0 then                  -- Altitude
             STOP_REASON:=HIT_GROUND;
             return true;
         elsif ENVIRONMENT.TIME >= 500.0 then
             STOP_REASON:=MAX_TIME_OF_FLIGHT;
             return true;
         else
             return false;
         end if;
     end END_CONDITIONS_MET;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
     procedure TERMINAL_CONDITIONS (A_POLE_OUT, MISS_DISTANCE_OUT,
                                    TIME_OF_FLIGHT, ALTITUDE, RDOT: out REAL;
                                    STOP_REASON_OUT: out STOP_CONDITION_TYPE) is
     TEMP: REAL;
     begin
         A_POLE_OUT := A_POLE;
         RDOT:=RDOT_OLD;
         STOP_REASON_OUT := STOP_REASON;
         if STOP_REASON = CROSSOVER then
             MISS_DISTANCE_OUT:=MISS_DISTANCE;
             TIME_OF_FLIGHT:=ENVIRONMENT.TIME-INTEGRATION.STEP_SIZE+
                             TIME_TO_GO_OLD;
             ALTITUDE:=ALTITUDE_OLD+((-MSL_POS_NED(3)-ALTITUDE_OLD)*
                        TIME_TO_GO_OLD/INTEGRATION.STEP_SIZE);
         else
             MISS_DISTANCE_OUT:=TGT_RANGE(1);
             TIME_OF_FLIGHT:=ENVIRONMENT.TIME;
             ALTITUDE:=-MSL_POS_NED(3);
         end if;
     end TERMINAL_CONDITIONS;

end MISSILE;

-------------------------------------------------------------
-- Airframe procedures
--
-- These procedures consist of routines for the aero
-- and thrust models.
```

```
--
-----------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;

package AIRFRAME is

    procedure INITIALIZE;

    procedure AERO(MSL_MACH,Q:     in REAL;
        RADOME_OFF:                    in boolean;
        COEF_DRAG:                   in out REAL;
        ANGLE_OF_ATTACK_OUT:         out VECTOR);

    procedure THRUST(MSL_MASS,THRUST:  out REAL;
                  PROPULSION_PHASE_OUT:  out PROPUL_TYPE);

end AIRFRAME;

-------------------------------------------------------------------------------
-- Airframe procedures
--
-- These procedures consist of routines for the aero and
-- thrust models.
--
-------------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;
with ENVIRONMENT;
with INTEGRATION;
with AUTOPILOT;

package body AIRFRAME is

    TIME_BOOSTER: constant :=  4.00;
    T_BOOST_INIT: constant :=  0.0; --1.0;-- time before booster ignited
    MSL_MASS_DRY: constant :=  300.0;
    ROCKET_MASS_INIT: constant := 100.0;
    PROPULSION_PHASE: PROPUL_TYPE;
    ACC_ACH:   VECTOR(2..3);
    BOOST_FUEL_RATE, EXP_DECAY: real;
    MSL_MASS, ROCKET_MASS: real;
    ANGLE_OF_ATTACK:  VECTOR(1..3);
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure INITIALIZE is
    begin
        MSL_MASS := MSL_MASS_DRY + ROCKET_MASS_INIT;
        PROPULSION_PHASE := COAST;    --initial coast before booster ignition
        BOOST_FUEL_RATE := ROCKET_MASS_INIT/(REAL(INTEGER
        ((TIME_BOOSTER/INTEGRATION.STEP_SIZE) + 0.5)) * INTEGRATION.STEP_SIZE);
    end INITIALIZE;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
    procedure AERO(MSL_MACH,Q:  in REAL;
        RADOME_OFF:                  in boolean;
        COEF_DRAG:                 in out REAL;
        ANGLE_OF_ATTACK_OUT:       out VECTOR)  is

        CD_ZERO,CD_INDUCED,AOA_ALPHA,
        AOA_BETA,AOA_TOTAL: REAL;

    begin
        -- compute AOA
        ACC_ACH := AUTOPILOT.ACCELERATIONS;
        AOA_ALPHA := -2.09 * ACC_ACH(3) / Q;
        if abs(AOA_ALPHA) > 0.5236 then
            AOA_ALPHA := SIGN(AOA_ALPHA) * 0.5236;
        end if;
        AOA_BETA := 2.09 * ACC_ACH(2) / Q;
        if abs(AOA_BETA) > 0.5236 then
            AOA_BETA := SIGN(AOA_BETA) * 0.5236;
        end if;
        AOA_TOTAL := SQRT(AOA_ALPHA * AOA_ALPHA + AOA_BETA * AOA_BETA);
        ANGLE_OF_ATTACK(1) := AOA_TOTAL;
        ANGLE_OF_ATTACK(2) := AOA_ALPHA;
        ANGLE_OF_ATTACK(3) := AOA_BETA;
        ANGLE_OF_ATTACK_OUT := ANGLE_OF_ATTACK;

        -- compute drag coefficient
        if MSL_MACH < 1.2 then
            CD_ZERO := 0.75 * MSL_MACH - 0.02;
        elsif MSL_MACH >= 1.2 and MSL_MACH < 1.8 then
            CD_ZERO := 2.346 - MSL_MACH * ((1.346-0.253 * MSL_MACH) * MSL_MACH-2.472);
        else
            CD_ZERO := 1.3 * EXP(-0.287 * MSL_MACH);
        end if;
        --10% increase after IR dome is removed
        if RADOME_OFF then
            CD_ZERO := 1.1 * CD_ZERO;
        end if;
        CD_INDUCED :=  7.0 * (AOA_TOTAL**2);
        COEF_DRAG := 0.97 * CD_ZERO + CD_INDUCED;

    end AERO;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure THRUST(MSL_MASS,THRUST:  out REAL;
                     PROPULSION_PHASE_OUT:  out PROPUL_TYPE)  is

        THRUST_BOOST: constant :=  11000.0;

    begin
        if ENVIRONMENT.Time < T_BOOST_INIT then
            PROPULSION_PHASE := COAST;
            THRUST := 0.0;
            MSL_MASS := MSL_MASS_DRY + ROCKET_MASS;
        elsif ENVIRONMENT.Time <= TIME_BOOSTER + T_BOOST_INIT then
            PROPULSION_PHASE := BOOST;
            THRUST := THRUST_BOOST * TIME_BOOSTER/(REAL(INTEGER((TIME_BOOSTER/
                        INTEGRATION.STEP_SIZE) - 0.5)) * INTEGRATION.STEP_SIZE);
            ROCKET_MASS := ROCKET_MASS_INIT - BOOST_FUEL_RATE * ENVIRONMENT.TIME;
            MSL_MASS := MSL_MASS_DRY + ROCKET_MASS;
        elsif ENVIRONMENT.Time > TIME_BOOSTER + T_BOOST_INIT then
            MSL_MASS := MSL_MASS_DRY;
            PROPULSION_PHASE := COAST;
            THRUST := 0.0;
        end if;
        PROPULSION_PHASE_OUT := PROPULSION_PHASE;
    end THRUST;

end AIRFRAME;

-------------------------------------------------------------
-- Autopilot Specification
--
-- These procedures consist of routines for modelling the autopilot.
--
--------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;
```

```
package AUTOPILOT is

    procedure INITIALIZE;

    procedure UPDATE_DIFF_EQS;

    procedure COMPUTE(ACC_COMMANDED: in VECTOR; ACC_ACHIEVED: out VECTOR);

    function ACCELERATIONS return VECTOR;

end AUTOPILOT;

--------------------------------------------------------------------------------
-- Autopilot Package
--
-- These procedures consist of routines for modelling the autopilot.
--
--------------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with ENVIRONMENT;
with INTEGRATION;
with REAL_MATRIX; use REAL_MATRIX;

package body AUTOPILOT is

    AP_NAT_FREQ : constant := 0.80; -- Hertz
    AP_DAMP_RATIO : constant := 0.75;
    AP_C1, AP_C2, AP_C3 : REAL;
    ACC_COM, ACC_ACH, ACC_ACH_2 : VECTOR(2..3);
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure INITIALIZE is
    begin
        AP_C1 := 2.0 * EXP(-AP_DAMP_RATIO * AP_NAT_FREQ * 2.0 * PI *
            INTEGRATION.STEP_SIZE) * COS(AP_NAT_FREQ * 2.0 * PI *
            INTEGRATION.STEP_SIZE * SQRT(1.0 - AP_DAMP_RATIO**2));

        AP_C2 := -EXP(-2.0 * AP_DAMP_RATIO * AP_NAT_FREQ * 2.0 * PI *
            INTEGRATION.STEP_SIZE);

        AP_C3 := 1.0 - AP_C1 - AP_C2;
        ACC_ACH   := (0.0,0.0);
        ACC_ACH_2 := (0.0,0.0);
    end INITIALIZE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure UPDATE_DIFF_EQS is
    begin
        ACC_ACH(2) := AP_C1*ACC_ACH(2) + AP_C2*ACC_ACH_2(2) + AP_C3*ACC_COM(2);
        ACC_ACH(3) := AP_C1*ACC_ACH(3) + AP_C2*ACC_ACH_2(3) + AP_C3*ACC_COM(3);
        ACC_ACH_2:=ACC_ACH;
    end UPDATE_DIFF_EQS;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure COMPUTE(ACC_COMMANDED: in VECTOR; ACC_ACHIEVED: out VECTOR) is
    begin
        ACC_COM:=ACC_COMMANDED;
        ACC_ACHIEVED:=ACC_ACH;
    end COMPUTE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function ACCELERATIONS return VECTOR is
    begin
         return ACC_ACH;
    end ACCELERATIONS;

end AUTOPILOT;
```

```
---------------------------------------------------------------------------
-- RF Seeker package
--
-- This package is the governing routine which estimates RF
-- acquistion times.  Seeker calculates a missile seeker gimbal angles.
-- It will be assumed that the antenna point directly along the LOS vector.
--
---------------------------------------------------------------------------
with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;

package RF_SEEKER is

    procedure SETUP(NUM_OF_TGTS_INIT,
                    NUM_OF_SOJS_INIT:    in INTEGER;
                    RCS_INIT:            in VECTOR;
                    ECM_POWER_INIT:      in VECTOR;
                    SSJ_ECM_TECH_INIT:   in SSJ_ECM_TECHNIQUE;
                     SOJ_ECM_TECH_INIT:    in SOJ_ECM_TECHNIQUE);

    procedure INITIALIZE(LAC_GAIN_IN,
                         LAC_TRANS_PWR_IN,
                          LAC_BEAMWIDTH_IN:  in REAL);

    procedure GIMBAL(RANGE_VEC_BOD_INIT:          in SUPER_VECTOR;
                     GIMBAL_ANGLE :     in out  VECTOR);

     procedure DETECTION(TGT_RANGE_IN,
                        RANGE_TGT_LAC_IN: in      VECTOR;
                        RF_PHASE_OUT:     out     RF_PHASE_TYPE2;
                        SNR_OUT:          out     REAL;
                         TGTTYP_OUT:       out       INTEGER;
                         BSE_OUT:           out      VECTOR);


end RF_SEEKER;

-----------------------------------------------------------------------------
-- RF_Seeker package
--
-- This package is the governing routine which estimates RF
-- acquistion times.  Seeker calculates a missile rf seeker gimbal angles.
-- It will be assumed that the antenna points directly along the LOS vector.
--
-----------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;
with ENVIRONMENT;

package body RF_SEEKER is

    RF_PHASE     : RF_PHASE_TYPE2;
    LAC_TYPE     : AIRCRAFT_TYPE;
    SEL_TGT      : GUID_SELECTION_TYPE;
    SSJ_ECM_TECH : SSJ_ECM_TECHNIQUE;
    SOJ_ECM_TECH : SOJ_ECM_TECHNIQUE;
    RCS          : VECTOR(1..2);
    ECM_POWER    : VECTOR(1..4);
    SNR          : REAL;
    K_ACQ_TIME   : REAL;
    X_ACQ_TIME   : REAL;
    LAC_TRANS_PWR: REAL;
    LAC_GAIN     : REAL;
    LAC_BEAMWIDTH: REAL;
    TGT_RANGE    : VECTOR(1..4);
    RANGE_TGT_LAC: VECTOR(1..4);
    NUM_OF_TGTS : INTEGER;
    NUM_OF_SOJS  : INTEGER;
    RANGE_VEC_BOD: SUPER_VECTOR(1..4);
    SELTGT       : INTEGER;
    TGTTYP        : INTEGER;
    BSE           : VECTOR(1..4);
    TGT_POWER     : VECTOR(1..2);
    REPEAT_POWER : VECTOR(1..2);
    NOISE_POWER  : VECTOR(1..4);
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure SETUP(NUM_OF_TGTS_INIT,
                 NUM_OF_SOJS_INIT:         in INTEGER;
                RCS_INIT:             in VECTOR;
                ECM_POWER_INIT:       in VECTOR;
                SSJ_ECM_TECH_INIT:    in SSJ_ECM_TECHNIQUE;
                 SOJ_ECM_TECH_INIT :   in SOJ_ECM_TECHNIQUE) is

begin
    RCS := RCS_INIT;
    SSJ_ECM_TECH := SSJ_ECM_TECH_INIT;
    SOJ_ECM_TECH := SOJ_ECM_TECH_INIT;
    ECM_POWER := ECM_POWER_INIT;
    NUM_OF_TGTS := NUM_OF_TGTS_INIT;
    NUM_OF_SOJS := NUM_OF_SOJS_INIT;
end SETUP;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure INITIALIZE(LAC_GAIN_IN, LAC_TRANS_PWR_IN,
                      LAC_BEAMWIDTH_IN:  in REAL) is
begin
    K_ACQ_TIME:= -10.0;
    X_ACQ_TIME:= -10.0;
    SNR:= -140.0;
    RF_PHASE:= NON_ACTIVE;
    LAC_GAIN:=LAC_GAIN_IN;
    LAC_TRANS_PWR:=LAC_TRANS_PWR_IN;
    LAC_BEAMWIDTH:=LAC_BEAMWIDTH_IN;
    TGT_POWER  := (-200.0,-200.0);
    REPEAT_POWER := (-200.0,-200.0);
    NOISE_POWER := (-200.0,-200.0,-200.0,-200.0);
    SELTGT:= 0;
    TGTTYP:= 0;
    BSE  := (0.0,0.0,0.0,0.0);
end INITIALIZE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure GIMBAL(RANGE_VEC_BOD_INIT : in SUPER_VECTOR;
                 GIMBAL_ANGLE : in out VECTOR) is

begin
    RANGE_VEC_BOD:= RANGE_VEC_BOD_INIT;
    if SELTGT = 0 then
        SELTGT := 1;
    end if;

    if RANGE_VEC_BOD(SELTGT)(1) = 0.0 then
        GIMBAL_ANGLE(3) := ATAN(RANGE_VEC_BOD(SELTGT)(2)/
                                RANGE_VEC_BOD(SELTGT)(3));
        GIMBAL_ANGLE(2) := ATAN(-RANGE_VEC_BOD(SELTGT)(3)/0.001);
        GIMBAL_ANGLE(1) := ACOS(COS(GIMBAL_ANGLE(2))*COS(GIMBAL_ANGLE(3)));
    else
        GIMBAL_ANGLE(3) := ATAN(RANGE_VEC_BOD(SELTGT)(2)/
                SQRT(RANGE_VEC_BOD(SELTGT)(1)**2+RANGE_VEC_BOD(SELTGT)(3)**2));
        GIMBAL_ANGLE(2) := ATAN(-RANGE_VEC_BOD(SELTGT)(3)/
                                RANGE_VEC_BOD(SELTGT)(1));
        GIMBAL_ANGLE(1) := ACOS(COS(GIMBAL_ANGLE(2))*COS(GIMBAL_ANGLE(3)));
    end if;
end GIMBAL;
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure BORESIGHT_ERROR  is
begin
    BSE(SELTGT) := GAUSSIAN(0.0,0.1);
    for I in SELTGT+1..NUM_OF_TGTS loop
         if RANGE_VEC_BOD(I)(1) = 0.0 then
             BSE(I) := 0.0;
         else
             BSE(I) := ACOS(DOT_PRODUCT(RANGE_VEC_BOD(SELTGT),RANGE_VEC_BOD(I))/
                         (MAGNITUDE(RANGE_VEC_BOD(SELTGT))*
                         MAGNITUDE(RANGE_VEC_BOD(I))));
         end if;
    end loop;
    -- Calculate the boresight error targets less than the selected target
    for I in 1..SELTGT-1 loop
         if RANGE_VEC_BOD(I)(1) = 0.0 then
             BSE(I) := 0.0;
         else
             BSE(I) := ACOS(DOT_PRODUCT(RANGE_VEC_BOD(SELTGT),RANGE_VEC_BOD(I))/
                         (MAGNITUDE(RANGE_VEC_BOD(SELTGT))*
                        MAGNITUDE(RANGE_VEC_BOD(I))));
         end if;
    end loop;
    for I in SELTGT+1..NUM_OF_SOJS loop
         if RANGE_VEC_BOD(I)(1) = 0.0 then
             BSE(I) := 0.0;
         else
             BSE(I) := ACOS(DOT_PRODUCT(RANGE_VEC_BOD(SELTGT),RANGE_VEC_BOD(I))/
                         (MAGNITUDE(RANGE_VEC_BOD(SELTGT))*
                         MAGNITUDE(RANGE_VEC_BOD(I))));
         end if;
    end loop;
    -- Calculate the boresight error targets less than the selected target
    for I in 1..SELTGT-1 loop
         if RANGE_VEC_BOD(I)(1) = 0.0 then
             BSE(I) := 0.0;
         else
             BSE(I) := ACOS(DOT_PRODUCT(RANGE_VEC_BOD(SELTGT),
                         RANGE_VEC_BOD(I+1))/(MAGNITUDE(RANGE_VEC_BOD(SELTGT))*
                         MAGNITUDE(RANGE_VEC_BOD(I+1))));
         end if;
    end loop;
end BORESIGHT_ERROR;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure MSL_ANT_GAINS_SA (BSE: in REAL; SUM_GAIN,DELTA_GAIN:  out REAL) is
  ABS_BSE: REAL;
  S1:      REAL;

begin
    ABS_BSE := ABS(BSE);
    if ABS_BSE <= 38.0 then
          S1 := 35.0*((ABS(COS(BSE*ABS_BSE)))**0.7)-13.0;
          if ABS_BSE > 30.0 then
             S1 := S1+ (ABS_BSE-30.0)*0.625;
          end if;
    else
        S1:=14.6*EXP(-(ABS_BSE-38.0)/30.0)-25.0;
    end if;
    SUM_GAIN:=S1;
    if ABS_BSE >= 20.0 then
        DELTA_GAIN:=31.0*SQRT(SIN(ABS_BSE*7.66))-13.0;
    else
        DELTA_GAIN:=7.82+25.0*EXP(-(ABS_BSE-20.0)/30.0)-25.0;
    end if;
end MSL_ANT_GAINS_SA;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
procedure MSL_ANT_GAINS_A (BSE: in REAL; SUM_GAIN,DELTA_GAIN: out REAL) is
  ABS_BSE: REAL;
  S1:      REAL;

begin
    ABS_BSE := ABS(BSE);
    if ABS_BSE <= 38.0 then
        S1 := 46.0*((ABS(COS(BSE*ABS_BSE)))**0.7)-13.0;
    if ABS_BSE > 30.0 then
        S1 := S1+ (ABS_BSE-30.0)*0.625;
    end if;
    else
        S1:=14.6*EXP(-(ABS_BSE-38.0)/30.0)-25.0;
    end if;
    SUM_GAIN:=S1;
    if ABS_BSE >= 20.0 then
        DELTA_GAIN:=31.0*SQRT(SIN(ABS_BSE*7.66))-13.0;
    else
        DELTA_GAIN:=7.82+25.0*EXP(-(ABS_BSE-20.0)/30.0)-25.0;
    end if;
end MSL_ANT_GAINS_A;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure SA_POWERS is
    LOOP_GAIN, ERPD, MSL_GAIN, DELTA_GAIN : REAL;

begin
    for I in 1..NUM_OF_TGTS loop
        MSL_ANT_GAINS_SA(BSE(I),MSL_GAIN,DELTA_GAIN);
        TGT_POWER(I) := -53.0+ LAC_TRANS_PWR+LAC_GAIN+MSL_GAIN+10.0*log(RCS(I))
                    -20.0*log(RANGE_TGT_LAC(I))
                     -20.0*log(TGT_RANGE(I));
        case SSJ_ECM_TECH(I) is
            when REPEATER =>
                 LOOP_GAIN := ECM_POWER(I);
                 REPEAT_POWER(I) := -84.0 + LAC_TRANS_PWR + LAC_GAIN +
                   MSL_GAIN+LOOP_GAIN-20.0*log(RANGE_TGT_LAC(I))-
                   20.0*log(TGT_RANGE(I));
            when BARRAGE_NOISE =>
                 ERPD := ECM_POWER(I);
                 NOISE_POWER(I) := -78.2 + LAC_TRANS_PWR + LAC_GAIN +
                                   MSL_GAIN + ERPD - 20.0*log(TGT_RANGE(I));
            when others =>
                 null;
        end case;
    end loop;

    if NUM_OF_SOJS /= 0 then
        for I in 1..NUM_OF_SOJS loop
            MSL_ANT_GAINS_SA(BSE(I+2),MSL_GAIN,DELTA_GAIN);

            case SOJ_ECM_TECH(I) is
                 when BARRAGE_NOISE =>
                      ERPD:=ECM_POWER(I+2);
                      NOISE_POWER(I+2):=-88.0+ERPD+MSL_GAIN-20.0*LOG(
                                     TGT_RANGE(I+2));
                 when others =>
                      NOISE_POWER(I+2):=-166.0;
            end case;
        end loop;
    end if;
end SA_POWERS;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure SA_DETECT is
    THERM_NOISE_FLOOR : constant:=-166.0; -- in db/filter
    FALSE_ALARM_THRESHOLD  :  constant:=6.0; -- in db
    NOISE_THRESHOLD        :  constant:=8.0; -- in db
```

```
begin
    SA_POWERS;
    if NUM_OF_TGTS = 1 then
        if  TGT_POWER(1) >= REPEAT_POWER(1) and TGT_POWER(1) >= NOISE_POWER(1)
        then
            SNR := TGT_POWER(1) - THERM_NOISE_FLOOR;
            if TGT_POWER(1) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                SELTGT := 1;
                TGTTYP := 3;
            else
                SELTGT := 0;
                TGTTYP := 0;
            end if;
        elsif REPEAT_POWER(1) > TGT_POWER(1) and
        REPEAT_POWER(1) > NOISE_POWER(1) then
            SNR := REPEAT_POWER(1) - THERM_NOISE_FLOOR;
            if REPEAT_POWER(1) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                SELTGT := 1;
                TGTTYP := 2;
            else
                SELTGT := 0;
                TGTTYP := 0;
            end if;
        else
            SNR := NOISE_POWER(1) - THERM_NOISE_FLOOR;
            if NOISE_POWER(2) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                SELTGT := 1;
                TGTTYP := 2;
            else
                SELTGT := 0;
                TGTTYP := 0;
            end if;
        end if;
    else
        if TGT_POWER(1) >= TGT_POWER(2) and TGT_POWER(1) >= REPEAT_POWER(2)
        and TGT_POWER(1) >= REPEAT_POWER(1) and TGT_POWER(1) >= NOISE_POWER(1)
        and TGT_POWER(1) >= NOISE_POWER(2) then
              SNR := TGT_POWER(1) - THERM_NOISE_FLOOR;
              if TGT_POWER(1) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                  SELTGT := 1;
                  TGTTYP := 3;
              else
                  SELTGT := 0;
                  TGTTYP := 0;
              end if;
        elsif TGT_POWER(2) > TGT_POWER(1) and TGT_POWER(2) >= REPEAT_POWER(2)
           and TGT_POWER(2) >= REPEAT_POWER(1)
           and TGT_POWER(2) >= NOISE_POWER(1)
           and TGT_POWER(2) >= NOISE_POWER(2) then
                SNR := TGT_POWER(2) - THERM_NOISE_FLOOR;
                if TGT_POWER(2) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                    SELTGT := 2;
                    TGTTYP := 3;
                else
                   SELTGT := 0;
                   TGTTYP := 0;
                end if;
        elsif REPEAT_POWER(2) > TGT_POWER(1) and REPEAT_POWER(2)>TGT_POWER(2)
            and REPEAT_POWER(2)>REPEAT_POWER(1) and REPEAT_POWER(2)>NOISE_POWER(1)
            and REPEAT_POWER(2) > NOISE_POWER(2) then
                SNR := REPEAT_POWER(2) - THERM_NOISE_FLOOR;
                if REPEAT_POWER(2) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                    SELTGT := 2;
                    TGTTYP := 2;
                else
                    SELTGT := 0;
                    TGTTYP := 0;
                end if;
        elsif REPEAT_POWER(1) > TGT_POWER(1) and REPEAT_POWER(1) > TGT_POWER(2)
```

```
           and REPEAT_POWER(1)>REPEAT_POWER(1) and REPEAT_POWER(1)>NOISE_POWER(1)
           and REPEAT_POWER(1) > NOISE_POWER(2) then
               SNR := REPEAT_POWER(1) - THERM_NOISE_FLOOR;
               if REPEAT_POWER(1) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                   SELTGT := 1;
                   TGTTYP := 2;
               else
                   SELTGT := 0;
                   TGTTYP := 0;
               end if;
        elsif  NOISE_POWER(1)>TGT_POWER(1) and NOISE_POWER(1)>TGT_POWER(2)
            and NOISE_POWER(1) > REPEAT_POWER(1)
            and NOISE_POWER(1) > REPEAT_POWER(2)
            and NOISE_POWER(1) > NOISE_POWER(2) then
                SNR := NOISE_POWER(1) - THERM_NOISE_FLOOR;
                if NOISE_POWER(1) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                    SELTGT := 1;
                    TGTTYP := 2;
                else
                    SELTGT := 0;
                    TGTTYP := 0;
                end if;
        else
            SNR := NOISE_POWER(2) - THERM_NOISE_FLOOR;
            if NOISE_POWER(2) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                SELTGT := 2;
                TGTTYP := 2;
            else
                SELTGT := 0;
                TGTTYP := 0;
            end if;
        end if;
    end if;

    -- determine if the soj power is detectable
    for I in 1..NUM_OF_SOJS loop

        case SOJ_ECM_TECH(I) is
            when BARRAGE_NOISE =>
                if TGT_POWER(I) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD
                and TGT_POWER(I) > NOISE_POWER(I) then
                        null;
                elsif NOISE_POWER(I) > THERM_NOISE_FLOOR +
                                            FALSE_ALARM_THRESHOLD then
                        SELTGT := 1;
                        TGTTYP := 1;
                        SNR := NOISE_POWER(I) - THERM_NOISE_FLOOR;
                else
                    SELTGT := 1;
                    TGTTYP := 0;
                        if TGT_POWER(I) > NOISE_POWER(I) then
                            SNR := TGT_POWER(I) - THERM_NOISE_FLOOR;
                        else
                            SNR := NOISE_POWER(I) - THERM_NOISE_FLOOR;
                        end if;
                end if;
            when others =>
                SELTGT := 1;
                TGTTYP := 0;
                SNR := TGT_POWER(I) - THERM_NOISE_FLOOR;
        end case;
    end loop;
end SA_DETECT;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure SA_MODE is

X_ACQ_DELAY : CONSTANT := 5.0; -- X band acq to track delay time
```

```
begin
    case RF_PHASE is
        when MIDCOURSE¦NON_ACTIVE =>
            if SELTGT = 1 or SELTGT = 2 then
                RF_PHASE:=X_BAND_ACQUISITION;
                if X_ACQ_TIME = -10.0 then
                    X_ACQ_TIME:=ENVIRONMENT.Time;
                end if;
            end if;
        when X_BAND_ACQUISITION =>
            if ENVIRONMENT.Time >= X_ACQ_TIME+X_ACQ_DELAY then
                RF_PHASE:=X_BAND_TRACK;
            end if;
        when others =>
            null;
    end case;
end SA_MODE;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
procedure A_POWERS is

THERM_NOISE_FLOOR : CONSTANT := -150.0; -- in db/filter
MSL_POWER : CONSTANT := 30.0;
LOOP_GAIN : REAL; -- in db
ERPD : REAL; -- in db
MSL_GAIN, DELTA_GAIN : REAL;

begin
    for I in 1..NUM_OF_TGTS loop
        MSL_ANT_GAINS_A(BSE(I),MSL_GAIN,DELTA_GAIN);
        TGT_POWER(I) := -63.45+ MSL_POWER+2.0*MSL_GAIN+10.0*LOG(RCS(I))-40.0*
                  log(TGT_RANGE(I));

        case SSJ_ECM_TECH(I) is
            when REPEATER =>
                LOOP_GAIN := ECM_POWER(I);
                REPEAT_POWER(I) := -105.0 +LAC_TRANS_PWR +LAC_GAIN +MSL_GAIN +
                                  LOOP_GAIN - 40.0*log(TGT_RANGE(I));
            when BARRAGE_NOISE =>
                ERPD := ECM_POWER(I);
                NOISE_POWER(I) := -88.65 + LAC_TRANS_PWR + LAC_GAIN + MSL_GAIN
                                + ERPD - 20.0*log(TGT_RANGE(I));
            when others =>
                  null;
        end case;
    end loop;
    if NUM_OF_SOJS /= 0 then
        for I in 1..NUM_OF_SOJS loop
            MSL_ANT_GAINS_A(BSE(I+2),MSL_GAIN,DELTA_GAIN);

            case SOJ_ECM_TECH(I) is
                when BARRAGE_NOISE =>
                    ERPD:=ECM_POWER(I+2);
                    NOISE_POWER(I+2):=-100.0+ERPD+MSL_GAIN-20.0*LOG(TGT_RANGE(I+2));
                when others =>
                    NOISE_POWER(I+2):= THERM_NOISE_FLOOR;
             end case;
        end loop;
    end if;
end A_POWERS;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
procedure A_DETECT is
THERM_NOISE_FLOOR : CONSTANT := -150.0;  -- in db/filter
                                          -- assume a 1000 hz/filter BW
FALSE_ALARM_THRESHOLD : CONSTANT := 6.0; -- in db
NOISE_THRESHOLD       : CONSTANT := 8.0; -- in db

begin
```

```
-- Calculate Power Levels
   A_POWERS;
   if NUM_OF_TGTS = 1 then
      if TGT_POWER(1) > REPEAT_POWER(1) and TGT_POWER(1) > NOISE_POWER(1) then
         SNR := TGT_POWER(1) - THERM_NOISE_FLOOR;
         if TGT_POWER(1) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
            SELTGT := 1;
            TGTTYP := 5;
         end if;
      elsif REPEAT_POWER(1)>TGT_POWER(1) and REPEAT_POWER(1)>NOISE_POWER(1) then
         SNR := REPEAT_POWER(1) - THERM_NOISE_FLOOR;
         if REPEAT_POWER(1) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
            SELTGT := 1;
            TGTTYP := 4;
         end if;
      else
         SNR := NOISE_POWER(1) - THERM_NOISE_FLOOR;
         if NOISE_POWER(2) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
            SELTGT := 1;
            TGTTYP := 4;
         end if;
      end if;
   else
      if TGT_POWER(1) > TGT_POWER(2) and TGT_POWER(1) > REPEAT_POWER(2) and
         TGT_POWER(1) > REPEAT_POWER(1) and TGT_POWER(1) > NOISE_POWER(1) and
         TGT_POWER(1) > NOISE_POWER(2) then
            SNR := TGT_POWER(1) - THERM_NOISE_FLOOR;
            if TGT_POWER(1) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
               SELTGT := 1;
               TGTTYP := 4;
            end if;
      elsif TGT_POWER(2) > TGT_POWER(1) and TGT_POWER(2) > REPEAT_POWER(2) and
         TGT_POWER(2) > REPEAT_POWER(1) and TGT_POWER(2) > NOISE_POWER(1) and
         TGT_POWER(2) > NOISE_POWER(2) then
            SNR := TGT_POWER(2) - THERM_NOISE_FLOOR;
            if TGT_POWER(2) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
               SELTGT := 2;
               TGTTYP := 5;
            end if;
      elsif REPEAT_POWER(2) > TGT_POWER(1) and REPEAT_POWER(2) >
         TGT_POWER(2) and REPEAT_POWER(2)>REPEAT_POWER(1) and
         REPEAT_POWER(2)>NOISE_POWER(1) and REPEAT_POWER(2) > NOISE_POWER(2)
         then
            SNR := REPEAT_POWER(2) - THERM_NOISE_FLOOR;
            if REPEAT_POWER(2) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
              SELTGT := 2;
              TGTTYP := 4;
         end if;
      elsif REPEAT_POWER(1)>TGT_POWER(1) and REPEAT_POWER(1)>TGT_POWER(2)
         and REPEAT_POWER(1)>REPEAT_POWER(1) and
         REPEAT_POWER(1)>NOISE_POWER(1) and REPEAT_POWER(1) > NOISE_POWER(2)
         then
            SNR := REPEAT_POWER(1) - THERM_NOISE_FLOOR;
            if REPEAT_POWER(1) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
               SELTGT := 1;
               TGTTYP := 4;
            end if;
      elsif  NOISE_POWER(1)>TGT_POWER(1) and NOISE_POWER(1)>TGT_POWER(2)
         and NOISE_POWER(1)>REPEAT_POWER(1) and NOISE_POWER(1) >
         NOISE_POWER(1) and NOISE_POWER(1) > NOISE_POWER(2) then
            SNR := NOISE_POWER(1) - THERM_NOISE_FLOOR;
            if NOISE_POWER(1) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
               SELTGT := 1;
               TGTTYP := 4;
            end if;
      else
         SNR := NOISE_POWER(2) - THERM_NOISE_FLOOR;
         if NOISE_POWER(2) >  THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
            SELTGT := 2;
```

```
                TGTTYP := 4;
            end if;
        end if;
    end if;

    -- determine if the soj power is detectable
    for I in 1..NUM_OF_SOJS loop

        case SOJ_ECM_TECH(I) is
                when BARRAGE_NOISE =>
                    if TGT_POWER(I) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD and
                       TGT_POWER(I) > NOISE_POWER(I) then
                           null;
                elsif NOISE_POWER(I) > THERM_NOISE_FLOOR + FALSE_ALARM_THRESHOLD then
                   SELTGT := 1;
                    TGTTYP := 1;
                   SNR := NOISE_POWER(I) - THERM_NOISE_FLOOR;
                else
                   SELTGT := 1;
                    TGTTYP := 0;
                   if TGT_POWER(I) > NOISE_POWER(I) then
                        SNR := TGT_POWER(I) - THERM_NOISE_FLOOR;
                    else
                        SNR := NOISE_POWER(I) - THERM_NOISE_FLOOR;
                    end if;
                end if;
            when others =>
                SELTGT := 1;
                TGTTYP := 0;
                SNR := TGT_POWER(I) - THERM_NOISE_FLOOR;
        end case;
    end loop;
end A_DETECT;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure A_MODE is
    K_ACQ_DELAY : CONSTANT:=  5.0;

begin
    case RF_PHASE is
        when X_BAND_TRACK¦NON_ACTIVE =>
            if  TGTTYP in 4..5 then
                RF_PHASE:=k_band_acquisition;
                if K_ACQ_TIME = -10.0 then
                    K_ACQ_TIME:=ENVIRONMENT.Time;
                end if;
            end if;
        when k_band_acquisition =>
            if ENVIRONMENT.Time >= K_ACQ_TIME+K_ACQ_DELAY then
                RF_PHASE:=K_BAND_TRACK;
            end if;
        when others =>
            null;
    end case;
end A_MODE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure DETECTION(TGT_RANGE_IN,
                    RANGE_TGT_LAC_IN     :  in  VECTOR;
                   RF_PHASE_OUT          :   out RF_PHASE_TYPE2;
                   SNR_OUT               :   out REAL;
                    TGTTYP_OUT           :   out INTEGER;
                    BSE_OUT              :   out VECTOR) is

begin
    TGT_RANGE:=TGT_RANGE_IN;
    RANGE_TGT_LAC:=RANGE_TGT_LAC_IN;
    -- Determine if we have aquired the target in S/A X-band, Ka band or are
    -- still in midcourse
    if RF_PHASE /= K_BAND_ACQUISITION or RF_PHASE /= K_BAND_TRACK then
        SA_DETECT;
        SA_MODE;
    end if;
    A_DETECT;
    A_MODE;
    SNR_OUT:=SNR;
    RF_PHASE_OUT:=RF_PHASE;
    BSE_OUT:=BSE;
    TGTTYP_OUT:=TGTTYP;
end DETECTION;

end RF_SEEKER;

-----------------------------------------------------------------------------
-- Package IR_SEEKER
--
--
-----------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;

package IR_SEEKER is

    procedure SETUP(IR_TGT1_SIGNATURE_IN, IR_TGT2_SIGNATURE_IN:
                    in TGT_IR_SIZE_TYPE);

    procedure INITIALIZE(TGT_ALTITUDE, MSL_ALTITUDE:  in REAL;
                         IR_ACQ_RANGE:  out REAL);

    procedure DETECTION (RANGE1 : in REAL; IR_PHASE_OUT : out
                         IR_PHASE_TYPE);

end IR_SEEKER;

-----------------------------------------------------------------------------
-- Package IR_SEEKER body
--
-----------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;
with ENVIRONMENT;

package body IR_SEEKER is

IR_PHASE: IR_PHASE_TYPE;
IR_SEARCH_TIME: REAL;
IR_ACQ_TIME   : REAL;
IR_TRK_TIME   : REAL;
IR_TGT1_SIGNATURE : TGT_IR_SIZE_TYPE;
IR_TGT2_SIGNATURE : TGT_IR_SIZE_TYPE;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
procedure SETUP(IR_TGT1_SIGNATURE_IN, IR_TGT2_SIGNATURE_IN:
                in TGT_IR_SIZE_TYPE) is
begin
    IR_TGT1_SIGNATURE:=IR_TGT1_SIGNATURE_IN;
    IR_TGT2_SIGNATURE:=IR_TGT2_SIGNATURE_IN;
    IR_PHASE:= NON_ACTIVE;
    IR_SEARCH_TIME:=0.0;
    IR_ACQ_TIME:=0.0;
    IR_TRK_TIME:=0.0;
end SETUP;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
```

```
procedure INITIALIZE(TGT_ALTITUDE, MSL_ALTITUDE:  in REAL;
                     IR_ACQ_RANGE:  out REAL) is
begin
    if TGT_ALTITUDE > 70_000.0 then
        if IR_TGT1_SIGNATURE = SMALL then
            IR_ACQ_RANGE:=200000.0;
        else
            IR_ACQ_RANGE:=400000.0;
        end if;
    elsif TGT_ALTITUDE > 10_000.0 then
        if IR_TGT1_SIGNATURE = SMALL then
            IR_ACQ_RANGE:=100000.0;
        elsif IR_TGT1_SIGNATURE = MEDIUM then
            IR_ACQ_RANGE:=200000.0;
        elsif IR_TGT1_SIGNATURE = LARGE then
            IR_ACQ_RANGE:=350000.0;
        end if;
    else
        if IR_TGT1_SIGNATURE = SMALL then
            IR_ACQ_RANGE:=6000.0;
        elsif IR_TGT1_SIGNATURE = MEDIUM then
            IR_ACQ_RANGE:=100000.0;
        elsif IR_TGT1_SIGNATURE = LARGE then
            IR_ACQ_RANGE:=150000.0;
        end if;
    end if;
end INITIALIZE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
procedure DETECTION (RANGE1 : in REAL; IR_PHASE_OUT : out
                     IR_PHASE_TYPE) is

       SEARCH_DELAY  : constant  := 5.0;  -- delay from radome off until IR_SEARCH
       ACQ_DELAY     : constant  := 1.0;  --       from IR_SEARCH until IR_ACQUISITION
       TRK_DELAY     : constant  := 1.0;  --       from IR_ACQUISITON until IR_TRACK

       DELTA_SEARCH  : REAL;     -- local variables
       DELTA_ACQ     : REAL;
       DELTA_TRK     : REAL;
begin
    case IR_PHASE is
    when NON_ACTIVE =>
        if IR_SEARCH_TIME = 0.0 then
            IR_SEARCH_TIME := ENVIRONMENT.Time;
        end if;
        DELTA_SEARCH := ENVIRONMENT.Time - IR_SEARCH_TIME;
        if DELTA_SEARCH >= SEARCH_DELAY then
            IR_PHASE:= IR_SEARCH;
        end if;
    when IR_SEARCH =>
        if IR_ACQ_TIME = 0.0 then
            IR_ACQ_TIME := ENVIRONMENT.Time;
        end if;
        DELTA_ACQ := ENVIRONMENT.Time - IR_ACQ_TIME;
        if DELTA_ACQ >= ACQ_DELAY then
            IR_PHASE:= IR_ACQUISITION;
        end if;
    when IR_ACQUISITION =>
        if IR_TRK_TIME = 0.0 then
            IR_TRK_TIME := ENVIRONMENT.Time;
        end if;
        DELTA_TRK := ENVIRONMENT.Time - IR_TRK_TIME;
        if DELTA_TRK >= TRK_DELAY then
            IR_PHASE:= IR_TRACK;
        end if;
    when IR_TRACK =>
            null;            -- Once in IR_TRACK always in IR_TRACK
    end case;
    IR_PHASE_OUT   := IR_PHASE;
    end DETECTION;

end IR_SEEKER;

--------------------------------------------------------------------------------
-- Guidance Package Specification
--
-- This package contains the data types and subprograms used in modeling the
-- guidance subsystem.
--
--------------------------------------------------------------------------------

with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;
with MODEL_TYPES; use MODEL_TYPES;

package GUIDANCE is
    procedure INITIALIZE (
        ALTITUDE  : in REAL;
        TGT_ALTITUDE  : in REAL;
        TGT_RANGE  : in VECTOR;
        TGT_VEL  : in VECTOR);

    procedure COMPUTE (
        TIME_TO_GO  : in REAL;
        ALTITUDE  : in REAL;
        ALTITUDE_RATE  : in REAL;
        VELOCITY  : in REAL;
        AXIAL_ACC  : in REAL;
        PITCH_BOD  : in REAL;
        TGT_RANGE  : in REAL;
        TGT_RANGE_RATE  : in REAL;
        SKR_GIMBAL_ANGLE : in VECTOR;
        OMEGA1_SKR  : in VECTOR;
        GUIDANCE_PHASE_OUT : out GUIDANCE_PHASE_TYPE;
        ACC_CMD_BOD  : out VECTOR);

end GUIDANCE;

--------------------------------------------------------------------------------
-- Guidance Package Body
--
-- This package contains the data types and subprograms used in modeling the
-- guidance subsystem.
--
--------------------------------------------------------------------------------

with MATH; use MATH;
with MODEL_TYPES; use MODEL_TYPES;
with REAL_MATRIX; use REAL_MATRIX;
with ENVIRONMENT;

package body GUIDANCE is
    ----------------------------------------------------------------------------
    --  Mode selection constants
    ----------------------------------------------------------------------------

    GUIDANCE_INITIATE_TIME  : constant :=  0.8; -- Sec
    TURNDOWN_GIMBAL_ANGLE : constant :=  40.0 * DEG_TO_RAD; -- Rad
    TRANS_CLIMB_ANGLE  : constant :=  20.0 * DEG_TO_RAD; -- Rad

    ----------------------------------------------------------------------------
    --  Altitude hold algorithm constants
    ----------------------------------------------------------------------------

    DESIRED_ALTITUDE  : constant :=  70000.0; -- Feet
    ALTITUDE_DELTA  : constant :=  50.0; -- Feet
    ALTITUDE_GAIN  : constant :=  0.1; -- Ft/Sec / Ft
    ALTITUDE_RATE_GAIN  : constant :=  0.3; -- Ft/Sec^2 / Ft/Sec
    ALTITUDE_ACC_LIMIT  : constant :=  160.0; -- Ft/Sec^2

    ------------------------------------------------------------------------
    --  Guidance algorithm constants
    ------------------------------------------------------------------------

    LOAD_BIAS_FACTOR  : constant := -5.0 * G; -- Ft/Sec^2
    ACCEL_LIMIT : constant := 35.0 * G; -- Ft/Sec^2
    CRUISE_ALPHA_EST : constant :=  10.0 * DEG_TO_RAD; -- Rad

    ------------------------------------------------------------------------
    --  Data definitions
    ------------------------------------------------------------------------

    GUIDANCE_PHASE : GUIDANCE_PHASE_TYPE;
    PREV_GUIDANCE_PHASE : GUIDANCE_PHASE_TYPE;
    COS_TOT_GIMBAL_ANG : REAL;
    COS_TURNDOWN_GIMBAL_ANGLE : REAL;
    CLIMB_ANGLE  : REAL;
    AXIAL_ACC_COMP : REAL;
    PIT_ACC_CMD  : REAL;
    YAW_ACC_CMD  : REAL;
    ALTITUDE_RATE_LIMIT : REAL;
    ALT_RATE_LIMIT_SET : boolean;
    LOS_BIAS : REAL;
    HOLD_PATH_TIME : REAL;
    TERMINAL_TGO_THRESHOLD : REAL;
    LONG_RANGE : boolean;
    ALT_SWITCH, ALT_RATE_SWITCH : REAL;
    PIT_ACC_COMP_ENABLED : boolean;
----------------------------------------------------------------------------
----------------------------------------------------------------------------
    procedure INITIALIZE (
        ALTITUDE  : in REAL;
        TGT_ALTITUDE  : in REAL;
        TGT_RANGE  : in VECTOR;
        TGT_VEL  : in VECTOR) is

        RANGE_HORIZ : REAL;
        TGT_VEL_HORIZ : REAL;
        REL_VEL_NOSE, REL_VEL_TAIL, REL_VEL : REAL;
        HORIZ_ASPECT_ANGLE : REAL;

        DT_SEP, DX_SEP, DT_HOLD, DX_HOLD : REAL;
        R_5G_BIAS, X_5G_BIAS, H_5G_BIAS, T_5G_BIAS : REAL;
        R_VAR_ARC, DX_VAR_ARC, DT_VAR_ARC : REAL;
        X_ALT_HOLD, T_ALT_HOLD : REAL;
        DX_TGT : REAL;
        COS_ASPECT_ANGLE, SIN_ASPECT_ANGLE : REAL;
        GIM_PITCHOVR_RNG, DELTA_HEIGHT : REAL;
        TGO_PITCHOVR_RNG, TGO_PITCHOVR_RNG_NOSE, TGO_PITCHOVR_RNG_TAIL : REAL;
        DISCRIMINANT : REAL;
        LOS_PITCHOVR_RNG, LOS_PITCHOVR_RNG_NOSE, LOS_PITCHOVR_RNG_TAIL : REAL;
        PITCHOVR_RNG_NOSE, PITCHOVR_RNG_TAIL : REAL;
        RNG_NOSE, RNG_TAIL, DELTA_RNG, RADIUS, THRESHOLD_RANGE : REAL;
    begin
        GUIDANCE_PHASE := NULL_COMMANDS;
        ALT_RATE_LIMIT_SET := false;
        PIT_ACC_COMP_ENABLED := false;
        COS_TURNDOWN_GIMBAL_ANGLE := COS(TURNDOWN_GIMBAL_ANGLE);

        -- LOS rate bias and time-to-go threshold based on tgt altitude

        if TGT_ALTITUDE <= 50000.0 then
            LOS_BIAS := 0.0025;
            TERMINAL_TGO_THRESHOLD := 40.0;
        else
            LOS_BIAS := 0.001;
            TERMINAL_TGO_THRESHOLD := 15.0 + TGT_ALTITUDE*0.0005;
```

```
end if;

-- Horizontal tgt range and relative velocity

RANGE_HORIZ := SQRT(TGT_RANGE(1)**2 + TGT_RANGE(2)**2);
TGT_VEL_HORIZ := SQRT(TGT_VEL(1)**2 + TGT_VEL(2)**2);

-- Altitude hold mode rel. vel. for nose & tail aspect

REL_VEL_NOSE := 3000.0 + TGT_VEL_HORIZ;
REL_VEL_TAIL := 3000.0 - TGT_VEL_HORIZ;

-- Horizontal aspect angle

SIN_ASPECT_ANGLE := (TGT_VEL(2)*TGT_RANGE(1) -
    TGT_VEL(1)*TGT_RANGE(2)) / (RANGE_HORIZ * TGT_VEL_HORIZ);
COS_ASPECT_ANGLE := (TGT_VEL(1)*TGT_RANGE(1) +
    TGT_VEL(2)*TGT_RANGE(2)) / (RANGE_HORIZ * TGT_VEL_HORIZ);

HORIZ_ASPECT_ANGLE := ATAN2(SIN_ASPECT_ANGLE, COS_ASPECT_ANGLE);

-- Time to hold before 5g bias climb

if TGT_ALTITUDE > 70000.0 then
    if ALTITUDE > 20000.0 then
        if MAGNITUDE(TGT_RANGE) < 110.0*FEET_PER_NMI then
            DT_HOLD := 20.0;
        elsif MAGNITUDE(TGT_RANGE) > 120.0*FEET_PER_NMI then
            DT_HOLD := 10.0;
        else
            DT_HOLD := 20.0 + (MAGNITUDE(TGT_RANGE)-110.0*FEET_PER_NMI)*
            (10.0-20.0) / ((120.0-110.0)*FEET_PER_NMI);
        end if;

        HOLD_PATH_TIME := ((TGT_ALTITUDE-DESIRED_ALTITUDE) /
            (90000.0-DESIRED_ALTITUDE)) * ((ALTITUDE-20000.0) /
            (30000.0-20000.0)) * 20.0;

        if HOLD_PATH_TIME > DT_HOLD then
            HOLD_PATH_TIME := DT_HOLD;
        end if;
    else
        HOLD_PATH_TIME := 0.8;
    end if;
else
    HOLD_PATH_TIME := 0.8;
end if;

-- Estimated missile travel, separation phase

DT_SEP := 0.8;
DX_SEP := 1000.0 * DT_SEP;

-- Estimated missile travel, flight path hold phase

DT_HOLD := HOLD_PATH_TIME - DT_SEP;
DX_HOLD := ((1000.0 + 2600.0) / 2.0) * DT_HOLD;

-- Estimated missile travel, 5g bias phase

R_5G_BIAS := (2600.0 ** 2) / (-LOAD_BIAS_FACTOR);
X_5G_BIAS := DX_SEP + DX_HOLD + R_5G_BIAS * SIN(TRANS_CLIMB_ANGLE);
H_5G_BIAS := ALTITUDE + R_5G_BIAS * (1.0 - COS(TRANS_CLIMB_ANGLE));
T_5G_BIAS := TRANS_CLIMB_ANGLE * 2600.0 / (-LOAD_BIAS_FACTOR) + DT_SEP +
DT_HOLD;

-- Estimated missile travel, variable arc phase

R_VAR_ARC := (DESIRED_ALTITUDE - H_5G_BIAS) /
```

```
(1.0 - COS(TRANS_CLIMB_ANGLE));
DX_VAR_ARC := R_VAR_ARC * SIN(TRANS_CLIMB_ANGLE);
DT_VAR_ARC := TRANS_CLIMB_ANGLE * R_VAR_ARC / ((2600.0 + 3000.0) / 2.0);

-- Estimated missile travel, altitude hold phase

X_ALT_HOLD := X_5G_BIAS + DX_VAR_ARC + 3000.0*20.0;
T_ALT_HOLD := T_5G_BIAS + DT_VAR_ARC + 20.0;

-- Estimated target travel

DX_TGT := TGT_VEL_HORIZ * T_ALT_HOLD;

-- Gimbal angle pitchover range

DELTA_HEIGHT := DESIRED_ALTITUDE - TGT_ALTITUDE;
GIM_PITCHOVR_RNG := abs(DELTA_HEIGHT) / TAN(TURNDOWN_GIMBAL_ANGLE -
    CRUISE_ALPHA_EST*SIGN(DELTA_HEIGHT));

-- Time to go pitchover range

REL_VEL := REL_VEL_NOSE;
for I in 1..2 loop
    TGO_PITCHOVR_RNG_NOSE := TGO_PITCHOVR_RNG;
    DISCRIMINANT := (REL_VEL*TERMINAL_TGO_THRESHOLD) ** 2 -
    4.0 * (DELTA_HEIGHT ** 2);

    if DISCRIMINANT >= 0.0 then
        TGO_PITCHOVR_RNG := (REL_VEL*TERMINAL_TGO_THRESHOLD +
            SQRT(DISCRIMINANT)) / 2.0;
    else
        TGO_PITCHOVR_RNG := 0.0;
    end if;

    -- LOS rate pitchover range

    LOS_PITCHOVR_RNG_NOSE := LOS_PITCHOVR_RNG;
    DISCRIMINANT := (REL_VEL / (0.5*DEG_TO_RAD)) - abs(DELTA_HEIGHT);
    if DISCRIMINANT > 0.0 then
        LOS_PITCHOVR_RNG := SQRT(abs(DELTA_HEIGHT) * DISCRIMINANT);
    else
        LOS_PITCHOVR_RNG := 0.0;
    end if;

    REL_VEL := REL_VEL_TAIL;
end loop;

if REL_VEL_TAIL > 0.0 then
    TGO_PITCHOVR_RNG_TAIL := TGO_PITCHOVR_RNG;
    LOS_PITCHOVR_RNG_TAIL := LOS_PITCHOVR_RNG;
else
    TGO_PITCHOVR_RNG_TAIL := 0.0;
    LOS_PITCHOVR_RNG_TAIL := 0.0;
end if;

PITCHOVR_RNG_NOSE := MAX3(GIM_PITCHOVR_RNG, TGO_PITCHOVR_RNG_NOSE,
    LOS_PITCHOVR_RNG_NOSE);
PITCHOVR_RNG_TAIL := MAX3(GIM_PITCHOVR_RNG, TGO_PITCHOVR_RNG_TAIL,
    LOS_PITCHOVR_RNG_TAIL);

-- Compute algorithm selection threshold range

RNG_NOSE := X_ALT_HOLD + PITCHOVR_RNG_NOSE + DX_TGT;
RNG_TAIL := X_ALT_HOLD + PITCHOVR_RNG_TAIL - DX_TGT;

DELTA_RNG := (RNG_NOSE - RNG_TAIL) / 2.0;
RADIUS := (RNG_NOSE + RNG_TAIL) / 2.0;

THRESHOLD_RANGE := SQRT(RADIUS**2 - (DELTA_RNG*SIN_ASPECT_ANGLE)**2) -
```

```
        DELTA_RNG*COS_ASPECT_ANGLE;

        -- Determine if 'long' or 'short' range flight & set parameters

        if RANGE_HORIZ < THRESHOLD_RANGE then
            LONG_RANGE := false;
            ALT_SWITCH := 60000.0;
            ALT_RATE_SWITCH := 600.0;
        else
            LONG_RANGE := true;
            ALT_SWITCH := 69000.0;
            ALT_RATE_SWITCH := 0.0;
        end if;
    end INITIALIZE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure COMPUTE (
        TIME_TO_GO  : in REAL;     -- Midcourse estimated time to go
        ALTITUDE  : in REAL;     -- Msl altitude above round earth
        ALTITUDE_RATE  : in REAL;     -- Derivative of altitude
        VELOCITY  : in REAL;     -- Velocity magnitude
        AXIAL_ACC  : in REAL;     -- INS measured axis 1 accel.
        PITCH_BOD  : in REAL;     -- Pitch relative to earth
        TGT_RANGE  : in REAL;     -- Estimated range to tgt
        TGT_RANGE_RATE  : in REAL;     -- Derivative of TGT_RANGE
        SKR_GIMBAL_ANGLE : in VECTOR;  -- Seeker LOS angles
        OMEGA1_SKR  : in VECTOR;  -- Seeker LOS angle rates
        GUIDANCE_PHASE_OUT : out GUIDANCE_PHASE_TYPE;
        ACC_CMD_BOD  : out VECTOR) is -- Body acceleration cmds
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
        procedure GUIDANCE_MODE is
            TOTAL_LOS_RATE : REAL;
            TOTAL_GIMBAL_ANG : REAL;
            VEL_LIMIT : REAL;
        begin
            PREV_GUIDANCE_PHASE := GUIDANCE_PHASE;

            COS_TOT_GIMBAL_ANG := COS(SKR_GIMBAL_ANGLE(2)) *
            COS(SKR_GIMBAL_ANGLE(3));
            TOTAL_GIMBAL_ANG := ACOS(COS_TOT_GIMBAL_ANG);
            TOTAL_LOS_RATE := SQRT(OMEGA1_SKR(2)**2 + OMEGA1_SKR(3)**2);

            if ENVIRONMENT.Time >= GUIDANCE_INITIATE_TIME then
                if PREV_GUIDANCE_PHASE = TERMINAL or
                TIME_TO_GO <= TERMINAL_TGO_THRESHOLD then
                    GUIDANCE_PHASE := TERMINAL;

                elsif PREV_GUIDANCE_PHASE = TURN_DOWN then
                    GUIDANCE_PHASE := TERMINAL; -- Incomplete in 6DOF

                elsif PREV_GUIDANCE_PHASE = ALTITUDE_HOLD then
                    if TOTAL_LOS_RATE >= 0.5*DEG_TO_RAD or
                    COS_TOT_GIMBAL_ANG <= COS_TURNDOWN_GIMBAL_ANGLE then
                        GUIDANCE_PHASE := TERMINAL;
                    else
                        GUIDANCE_PHASE := ALTITUDE_HOLD;
                    end if;

                elsif PREV_GUIDANCE_PHASE = VARIABLE_ARC then
                    if TOTAL_LOS_RATE >= 0.5*DEG_TO_RAD then
                        GUIDANCE_PHASE := TERMINAL;
                    elsif ALTITUDE >= ALT_SWITCH and
                    ALTITUDE_RATE >= ALT_RATE_SWITCH then
                        GUIDANCE_PHASE := ALTITUDE_HOLD;
                    else
                        GUIDANCE_PHASE := VARIABLE_ARC;
                    end if;
            elsif PREV_GUIDANCE_PHASE = LOAD_BIAS then
                CLIMB_ANGLE := ASIN(ALTITUDE_RATE / VELOCITY);

                if CLIMB_ANGLE >= TRANS_CLIMB_ANGLE then
                    GUIDANCE_PHASE := VARIABLE_ARC;
                else
                    GUIDANCE_PHASE := LOAD_BIAS;
                end if;

            elsif PREV_GUIDANCE_PHASE = HOLD_PATH then
                if ALTITUDE <= 30000.0 then
                    VEL_LIMIT := 2900.0 + 0.0225*ALTITUDE;
                elsif ALTITUDE <= 70000.0 then
                    VEL_LIMIT := 3500.0 + 0.00725*(ALTITUDE-30000.0);
                else
                    VEL_LIMIT := 3800.0;
                end if;

                if VELOCITY >= (VEL_LIMIT-150.0) then
                    GUIDANCE_PHASE := LOAD_BIAS;
                elsif ENVIRONMENT.Time > HOLD_PATH_TIME then
                    GUIDANCE_PHASE := LOAD_BIAS;
                else
                    GUIDANCE_PHASE := HOLD_PATH;
                end if;

            elsif PREV_GUIDANCE_PHASE = NULL_COMMANDS then
                if HOLD_PATH_TIME <= 0.8 then
                    GUIDANCE_PHASE := LOAD_BIAS;
                else
                    GUIDANCE_PHASE := HOLD_PATH;
                end if;
            end if;
        end if;
    end GUIDANCE_MODE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function ALTITUDE_HOLD_CMD
        return REAL is

        ALTITUDE_RATE_CMD : REAL;
        ALTITUDE_ACC_CMD : REAL;
    begin
        if not ALT_RATE_LIMIT_SET then
            ALTITUDE_RATE_LIMIT := abs(ALTITUDE_RATE);
            ALT_RATE_LIMIT_SET := true;
        end if;

        ALTITUDE_RATE_CMD := -ALTITUDE_GAIN * (ALTITUDE - DESIRED_ALTITUDE);
        ALTITUDE_RATE_CMD :=  LIMIT(ALTITUDE_RATE_CMD, ALTITUDE_RATE_LIMIT);

        ALTITUDE_ACC_CMD := ALTITUDE_RATE_GAIN * (ALTITUDE_RATE -
            ALTITUDE_RATE_CMD);
        ALTITUDE_ACC_CMD := LIMIT(ALTITUDE_ACC_CMD, ALTITUDE_ACC_LIMIT);

        return ALTITUDE_ACC_CMD;
    end ALTITUDE_HOLD_CMD;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure GUIDANCE_COMMANDS is
        TOTAL_ACC_CMD : REAL;
        GUIDANCE_GAIN : REAL;
    begin
        if GUIDANCE_PHASE = TERMINAL then
            GUIDANCE_GAIN := 3.0;
            PIT_ACC_COMP_ENABLED := true;
        else
            GUIDANCE_GAIN := 4.0;
        end if;
```

```
-- Yaw acceleration command

case GUIDANCE_PHASE is
    when NULL_COMMANDS =>
        YAW_ACC_CMD := 0.0;
    when others =>
        AXIAL_ACC_COMP := AXIAL_ACC - G*SIN(PITCH_BOD);

        -- axial_acc_comp used for pitch command also

        YAW_ACC_CMD := -GUIDANCE_GAIN * TGT_RANGE_RATE *
        (OMEGA1_SKR(2) * TAN(SKR_GIMBAL_ANGLE(2)) *
            TAN(SKR_GIMBAL_ANGLE(3)) + OMEGA1_SKR(3) /
            COS(SKR_GIMBAL_ANGLE(3)));

        YAW_ACC_CMD := YAW_ACC_CMD + AXIAL_ACC_COMP *
        TAN(SKR_GIMBAL_ANGLE(3)) / COS(SKR_GIMBAL_ANGLE(2));
end case;

-- Prop. Nav. Pitch acceleration command

PIT_ACC_CMD := GUIDANCE_GAIN * TGT_RANGE_RATE *
OMEGA1_SKR(2) / COS(SKR_GIMBAL_ANGLE(2));

if PIT_ACC_COMP_ENABLED then
    PIT_ACC_CMD := PIT_ACC_CMD - AXIAL_ACC_COMP *
    TAN(SKR_GIMBAL_ANGLE(2));
end if;

-- Pitch acceleration command

case GUIDANCE_PHASE is
    when NULL_COMMANDS =>
        PIT_ACC_CMD := 0.0;

    when HOLD_PATH =>
        PIT_ACC_CMD := 0.0;

    when LOAD_BIAS =>
        if LONG_RANGE then
            PIT_ACC_CMD := LOAD_BIAS_FACTOR;
        else
            PIT_ACC_CMD := PIT_ACC_CMD + LOAD_BIAS_FACTOR;
        end if;

    when VARIABLE_ARC =>
        if LONG_RANGE then
            CLIMB_ANGLE := ASIN(ALTITUDE_RATE / VELOCITY);
            PIT_ACC_CMD := (1.0 - COS(CLIMB_ANGLE)) *
            (VELOCITY ** 2) / ((DESIRED_ALTITUDE + ALTITUDE_DELTA)
                - ALTITUDE);
        else
            PIT_ACC_CMD := PIT_ACC_CMD + GUIDANCE_GAIN *
            TGT_RANGE_RATE * LOS_BIAS / COS(PITCH_BOD);
        end if;

    when ALTITUDE_HOLD =>
        PIT_ACC_CMD := ALTITUDE_HOLD_CMD / COS(PITCH_BOD);

    when TURN_DOWN =>
        null; -- Use Prop. Nav. command

    when TERMINAL =>
        null; -- Use Prop. Nav. command
end case;

--     Compensate for gravity

PIT_ACC_CMD := PIT_ACC_CMD - G * COS(PITCH_BOD);
```

```
            --     Limit acceleration commands

            TOTAL_ACC_CMD := SQRT(YAW_ACC_CMD**2 + PIT_ACC_CMD**2);

            if TOTAL_ACC_CMD > ACCEL_LIMIT then
                PIT_ACC_CMD := ACCEL_LIMIT*(PIT_ACC_CMD / TOTAL_ACC_CMD);
                YAW_ACC_CMD := ACCEL_LIMIT*(YAW_ACC_CMD / TOTAL_ACC_CMD);
            end if;

        end GUIDANCE_COMMANDS;

    begin
        GUIDANCE_MODE;
        GUIDANCE_COMMANDS;

        GUIDANCE_PHASE_OUT := GUIDANCE_PHASE;
        ACC_CMD_BOD(2) := YAW_ACC_CMD;
        ACC_CMD_BOD(3) := PIT_ACC_CMD;
    end COMPUTE;
end GUIDANCE;

----------------------------------------------------------------------
-- Kinematics package
--
-- This package contains the procedures which will initialize
-- and then perform the necessary calculations to determine the kinematic
-- variables that are required to perform the simulation.
--
----------------------------------------------------------------------
with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;
with MODEL_TYPES; use MODEL_TYPES;

package KINEMATICS is

    procedure SETUP(NTGTS_IN,NSOJS_IN:  in INTEGER);

    procedure COMPUTE(MSL_POS, MSL_VEL, TGT_VEL:  in VECTOR; TGT_POS:
                      in SUPER_VECTOR; RANGE_VEC: in out SUPER_VECTOR;
                      OMEGA:  out VECTOR; TGT_RANGE, RDOT, MISS_DISTANCE_NED:
                      in out VECTOR; TIME_TO_GO, MISS_DISTANCE:  in out REAL);

    procedure EOM(MSL_POS, MSL_VEL_VEC, ANGLE_O_ATTACK:  in VECTOR; COEF_DRAG,
        MSL_MASS, THRUST:  in REAL; MSL_X_ACC_BOD:  out REAL;
        MSL_VEL, MSL_HEADING_AZ, PITCH, Q:  in out REAL);

    procedure MACH_NO(ALTITUDE:  in REAL; MISSILE_VEL:  in VECTOR;
        MACH:  out REAL);

    procedure DIR_COS(PSI, THETA:  in REAL; EIB:  out MATRIX);

end KINEMATICS;

-----------------------------------------------------------------------------
-- Kinematics package
--
-- This package contains the procedures which will initialize
-- and then perform the necessary calculations to determine the kinematic
-- variables that are required to perform the simulation.
--
-----------------------------------------------------------------------------

with ENVIRONMENT;
with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;

package body KINEMATICS is
    -- List of variables that are global to the KINEMATICS package.
    SREF    :   constant REAL:= 0.492;   -- Missile reference area, ft**2
    NTGTS        :        INTEGER;
    NSOJS        :        INTEGER;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SETUP(NTGTS_IN, NSOJS_IN:  in INTEGER) is
    begin
        NTGTS:=NTGTS_IN;
        NSOJS:=NSOJS_IN;
    end SETUP;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure COMPUTE(MSL_POS, MSL_VEL, TGT_VEL:  in VECTOR;
                      TGT_POS:  in SUPER_VECTOR; RANGE_VEC:
                      in out SUPER_VECTOR; OMEGA:  out VECTOR;
                      TGT_RANGE, RDOT, MISS_DISTANCE_NED:  in out VECTOR;
                      TIME_TO_GO, MISS_DISTANCE:  in out REAL) is

        RHO:     SUPER_VECTOR(1..4); --  Missile to target  vectors unit vector
        MSL_TGT_VEL: --  Missile to target one velocity vector
        VECTOR(1..3);

    begin
        -- Calculate MSL_TGT_VEL
        MSL_TGT_VEL:=TGT_VEL-MSL_VEL;

    for I in 1..NTGTS loop
        -- Calculate the missile to target range vector
        RANGE_VEC(I):=TGT_POS(I)-MSL_POS;
        -- Calculate the missile to target range
        TGT_RANGE(I):=MAGNITUDE(RANGE_VEC(I));
        -- Calculate the unit vector of the range vector
        RHO(I):=RANGE_VEC(I)/TGT_RANGE(I);
        -- Calculate RDOT(I)
        RDOT(I):= DOT_PRODUCT(RHO(I),MSL_TGT_VEL);
    end loop;

    for I in 1..NSOJS loop
        RANGE_VEC(I+2):=TGT_POS(I+2)-MSL_POS;
        TGT_RANGE(I+2):=MAGNITUDE(RANGE_VEC(I+2));
    end loop;

    -- Calculate the Line of sight (LOS) vector OMEGA
    OMEGA:=CROSS_PRODUCT(RHO(1),MSL_TGT_VEL)/TGT_RANGE(1);
    -- Calculate the estimate of the time to go
    if RDOT(1) = 0.0 then
        RDOT(1):=0.001;
    end if;

    MISS_DISTANCE_NED:=RANGE_VEC(1)-MSL_TGT_VEL*
                       DOT_PRODUCT(RANGE_VEC(1),MSL_TGT_VEL)
                       /MAGNITUDE(MSL_TGT_VEL)/MAGNITUDE(MSL_TGT_VEL);
    MISS_DISTANCE:=MAGNITUDE(MISS_DISTANCE_NED);

    if TIME_TO_GO <= 4.0 then
        TIME_TO_GO:=(-SQRT(TGT_RANGE(1)*TGT_RANGE(1)-MISS_DISTANCE*
                MISS_DISTANCE))/RDOT(1);
    else
        TIME_TO_GO:=-TGT_RANGE(1)/RDOT(1);
    end if;

    if TIME_TO_GO >= 1000.0 or (ENVIRONMENT.TIME < 2.0 and
                          TIME_TO_GO < 0.0) then
        TIME_TO_GO:=1000.0;
    elsif TIME_TO_GO < 0.0 then
        TIME_TO_GO:=0.0;
    end if;
    end COMPUTE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
    procedure EOM(MSL_POS, MSL_VEL_VEC, ANGLE_O_ATTACK: in VECTOR; COEF_DRAG,
        MSL_MASS, THRUST: in REAL; MSL_X_ACC_BOD:  out REAL;
        MSL_VEL, MSL_HEADING_AZ, PITCH, Q:  in out REAL) is

        -- This procedure calculates the missile axial acceleration
        -- vector from the inputs.  This vector is returned to the
        -- MISSILE.COMPUTE procedure, where it is made available to
        -- the applications package for integration.
        DRAG : REAL;

    begin
        -- Calculate the missile heading angles
        PITCH:=ANGLE_O_ATTACK(2)+ATAN(-MSL_VEL_VEC(3)/SQRT(MSL_VEL_VEC(1)**2.0
                +MSL_VEL_VEC(2)**2.0));
        MSL_HEADING_AZ:=ATAN(MSL_VEL_VEC(2)/MSL_VEL_VEC(1));
        -- Determine the missile velocity
        MSL_VEL:=MAGNITUDE(MSL_VEL_VEC);
        -- Calculate the dynamic pressure
        Q:=0.5*ENVIRONMENT.AIR_DENSITY(-MSL_POS(3))*MSL_VEL*MSL_VEL;
        -- Calculate the drag on the missile
        DRAG:=Q*COEF_DRAG*SREF;
        -- Calculate the missile x acceleration
        MSL_X_ACC_BOD:= (THRUST-DRAG)*G/MSL_MASS;
    end EOM;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure MACH_NO(ALTITUDE:  in REAL; MISSILE_VEL:  in VECTOR;
        MACH:  out REAL) is
    begin
        MACH:=MAGNITUDE(MISSILE_VEL)/ENVIRONMENT.SPEED_OF_SOUND(ALTITUDE);
    end MACH_NO;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure DIR_COS(PSI,THETA: in REAL; EIB: out MATRIX) is
        CPSI, --  Cos(PSI)
        SPSI, --  Sin(PSI)
        CTHE, --  Cos(THETA)
        STHE: --  Sin(THETA)
        REAL;

    begin
        CPSI:= COS(PSI);
        SPSI:= SIN(PSI);
        CTHE:= COS(THETA);
        STHE:= SIN(THETA);
        EIB(1,1):= CTHE*CPSI;
        EIB(1,2):= CTHE*SPSI;
        EIB(1,3):= -STHE;
        EIB(2,1):= -SPSI;
        EIB(2,2):=  CPSI;
        EIB(2,3):=  0.0;
        EIB(3,1):= STHE*CPSI;
        EIB(3,2):= STHE*SPSI;
        EIB(3,3):= CTHE;
    end DIR_COS;

end KINEMATICS;

------------------------------------------------------------------------
-- Targets package
--
-- This package contains the procedure calls which will initialize,
-- setup, computes and get and put the missile state vector as well
-- as calculating the necessary parameters to calculate the state
-- vector.
-----------------------------------------------------------------------
with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;
with MODEL_TYPES; use MODEL_TYPES;
```

```
package TARGETS is

    procedure SETUP(ASPECT_IN,
                    TGT2_ANGLE_IN,
                    TGT_MACH_IN,
                    NO_OF_GS_IN,
                    WEAVE_PERIOD_IN,
                    TURN_ON_VALUE_IN,
                    TURN_ANGLE_IN,
                    BUILDUP_TIME_IN:  in REAL;
                    TGT_RANGE_IN,
                    TGT_ALT_IN,
                    SOJ_ANGLE_IN:  in VECTOR;
                    MANEUVER_IN:  in MANEUVER_TYPE;
                    TGT_TWO_IN,
                    SOJ_ONE_IN,
                    SOJ_TWO_IN:   in YES_NO_TYPE;
                    TURN_ON_PARAMETER_IN:  in MANEUVER_START_TYPE);

    procedure PUT_STATES(STATES:  in VECTOR);

    function GET_DERIVATIVES return VECTOR;

    function GET_STATES return VECTOR;

    function LOG_DATA return TARGET_LOG_DATA_TYPE;

    procedure INITIALIZE;

    procedure TGT_DATA(TGT_VEL_OUT:  out VECTOR;
                       TGT_POS_OUT:  out SUPER_VECTOR);

    function TGT_POS return SUPER_VECTOR;

    procedure COMPUTE;

    function TGT_ASPECT(RANGE_VEC:  in VECTOR) return REAL;

end TARGETS;

--------------------------------------------------------------------------------
-- Targets package
--
-- This package contains the procedure calls which will initialize,
-- setup, computes and get and put the missile state vector as well
-- as calculating the necessary parameters to calculate the state
-- vector.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

with MODEL_TYPES; use MODEL_TYPES;
with MATH; use MATH;
with ENVIRONMENT;
with REAL_MATRIX; use REAL_MATRIX;
with LAUNCHER;
with MISSILE;
WITH TEXT_IO;WITH REAL_IO;

package body TARGETS is

    LOGGED_DATA     :   TARGET_LOG_DATA_TYPE ;  -- Output Vector
    ASPECT          :   VECTOR(1..2);   -- Target Aspects
    TGT2_ANGLE       :   REAL;             -- Relative angle of tgt 2 to tgt1
    TGT_MACH        :   REAL;             -- Target Machs
    TGT_HEAD_DOT    :   REAL;           -- Target one heading angle rate
    TGT_HEAD_ANGLE  :   REAL;           -- Target one heading angle
    TGT_VEL_NED     :   VECTOR (1..3);  -- Target Velocity Vector
    TGT_POS_NED     :   SUPER_VECTOR (1..4);  -- Target Position Vectors
    TGT_VEL         :   REAL;           -- Magnitude of Target Velocity
    NO_OF_GS         :   REAL;           -- Number of g's pulled in maneuver
    WEAVE_PERIOD     :   REAL;           -- Period of target weave
    TURN_ON_VALUE    :   REAL;           -- Time maneuver begins
    TURN_ANGLE       :   REAL;           -- Angle turned through (turn and run)
    BUILDUP_TIME     :   REAL;    -- Exponential buildup time
    TGT_RANGE        :   VECTOR(1..4);   -- LOS Range to the targets
    SOJ_ANGLE         :   VECTOR(1..2);   -- Angle to SOJS relative to LAC az
    MANEUVER         :   MANEUVER_TYPE;  -- Type of maneuver
    TURN_ON_PARAMETER:   MANEUVER_START_TYPE;
    NSOJS             :   INTEGER;         -- Number of stand off jammers
    NTGTS            :   INTEGER;        -- Number of Targets
    DERIVATIVES      :   VECTOR(1..6);  --
    STATE            :   VECTOR(1..6);  --
    G_HORZ           :   REAL;          -- Horizontal g's pulled in weave
    OMEGA            :   REAL;          -- Freqency of the weave
    FSTART_MANEUVER : boolean;       -- Maneuver start flag
    START_TIME     : REAL;        -- Maneuver start time
    FINAL_HEAD_ANGLE: REAL;        -- Final heading angle after turn
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SETUP(ASPECT_IN,
                TGT2_ANGLE_IN,
                TGT_MACH_IN,
                NO_OF_GS_IN,
                WEAVE_PERIOD_IN,
                TURN_ON_VALUE_IN,
                TURN_ANGLE_IN,
                BUILDUP_TIME_IN:  in REAL;
                TGT_RANGE_IN,
                TGT_ALT_IN,
                SOJ_ANGLE_IN:  in VECTOR;
                MANEUVER_IN:  in MANEUVER_TYPE;
                TGT_TWO_IN,
                SOJ_ONE_IN,
                SOJ_TWO_IN:   in YES_NO_TYPE;
               TURN_ON_PARAMETER_IN:  in MANEUVER_START_TYPE) is
    TEMP: REAL;
    begin
        ASPECT(1) :=  DEG_TO_RAD*REAL(integer(ASPECT_IN*RAD_TO_DEG) mod 360);
        -- Initial Target 1 Aspect (rad)
         TGT2_ANGLE:=TGT2_ANGLE_IN;        -- Tgt 2 angle relative to tgt 1
        TGT_MACH :=  TGT_MACH_IN;   -- Initial Target Mach
        NO_OF_GS  :=  NO_OF_GS_IN;
        WEAVE_PERIOD:=WEAVE_PERIOD_IN;
         TURN_ON_VALUE:=TURN_ON_VALUE_IN;
        TURN_ANGLE:=  TURN_ANGLE_IN;
        BUILDUP_TIME:=BUILDUP_TIME_IN;
        TGT_RANGE    :=  TGT_RANGE_IN;   -- LOS Range (feet)
         SOJ_ANGLE := SOJ_ANGLE_IN;
        MANEUVER  :=  MANEUVER_IN;
         if TGT_TWO_IN = NO then
             NTGTS:=1;
         else
             NTGTS:=2;
         end if;
         if SOJ_ONE_IN = NO and SOJ_TWO_IN = NO then
             NSOJS:=0;
         elsif SOJ_ONE_IN = YES and SOJ_TWO_IN = NO then
             NSOJS:=1;
         elsif SOJ_ONE_IN = NO and SOJ_TWO_IN = YES then
             NSOJS:=1;
             TGT_RANGE(3):=TGT_RANGE(4);
             SOJ_ANGLE(1):=SOJ_ANGLE(2);
         elsif SOJ_ONE_IN = YES and SOJ_TWO_IN = YES then
             NSOJS:=2;
         end if;
        TURN_ON_PARAMETER:=TURN_ON_PARAMETER_IN;
         for I in 1..4 loop
             TGT_POS_NED(I)(3):=-TGT_ALT_IN(I);
        end loop;
    end SETUP;
--------------------------------------------------------------------------------
---------------------------------------------------------------------------
    procedure PUT_STATES(STATES: in VECTOR) is

    begin
        TGT_POS_NED(1):=STATES(1..3);
    end PUT_STATES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function GET_DERIVATIVES return VECTOR is

    begin
        DERIVATIVES(1..3):=TGT_VEL_NED;
        return DERIVATIVES;
    end GET_DERIVATIVES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function GET_STATES return VECTOR is

    begin
        STATE(1..3):=TGT_POS_NED(1);
        return STATE;
    end GET_STATES;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function LOG_DATA return TARGET_LOG_DATA_TYPE is

    begin
        LOGGED_DATA.REAL_VALUE(1):=  TGT_POS_NED(1)(1);
        LOGGED_DATA.REAL_VALUE(2):=  TGT_POS_NED(1)(2);
        LOGGED_DATA.REAL_VALUE(3):= -TGT_POS_NED(1)(3);
        LOGGED_DATA.REAL_VALUE(4):=  TGT_POS_NED(2)(1);
        LOGGED_DATA.REAL_VALUE(5):=  TGT_POS_NED(2)(2);
        LOGGED_DATA.REAL_VALUE(6):= -TGT_POS_NED(2)(3);
        LOGGED_DATA.REAL_VALUE(7):= TGT_VEL;
        LOGGED_DATA.REAL_VALUE(8):= TGT_MACH;
        LOGGED_DATA.REAL_VALUE(9):= TGT_HEAD_ANGLE*RAD_TO_DEG;
        return LOGGED_DATA;
    end LOG_DATA;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
        procedure INITIALIZE is
        TGT2_TEMP_ANGLE:          REAL;
        LAC_POS_NED : VECTOR(1..3);

        begin
            START_TIME:=0.0;
            FSTART_MANEUVER:=false;
            LAC_POS_NED:=LAUNCHER.LAC_POS;
            TGT_VEL:=TGT_MACH*ENVIRONMENT.SPEED_OF_SOUND(-TGT_POS_NED(1)(3));
            TGT_VEL_NED(1):=TGT_VEL*COS(ASPECT(1));
            TGT_VEL_NED(2):=TGT_VEL*SIN(ASPECT(1));
            TGT_VEL_NED(3):=0.0;
            TGT_POS_NED(1)(1):=SQRT(TGT_RANGE(1)*TGT_RANGE(1)-(-LAC_POS_NED(3)+
                TGT_POS_NED(1)(3))*(-LAC_POS_NED(3)+TGT_POS_NED(1)(3)));
            TGT_POS_NED(1)(2):=0.0;
            TGT_HEAD_DOT:=0.0;
            TGT_HEAD_ANGLE:=ASPECT(1);
            TGT2_TEMP_ANGLE:=TGT2_ANGLE+TGT_HEAD_ANGLE-PI;
            if NTGTS = 2 then
                TGT_POS_NED(2)(1):=TGT_POS_NED(1)(1)+SIN(TGT2_TEMP_ANGLE)*
                                   TGT_RANGE(2);
                TGT_POS_NED(2)(2):=TGT_POS_NED(1)(2)-COS(TGT2_TEMP_ANGLE)*
                                   TGT_RANGE(2);
                if TGT_POS_NED(2)(2) /= 0.0 then
                    ASPECT(2):=DEG_TO_RAD*REAL(INTEGER(RAD_TO_DEG*(ASPECT(1)-
                               ATAN(TGT_POS_NED(2)(2)/TGT_POS_NED(2)(1))))
```

```
                         mod 360);
           else
               ASPECT(2):=ASPECT(1);
           end if;
       else
           TGT_POS_NED(2)(1):=0.0;
           TGT_POS_NED(2)(2):=0.0;
           ASPECT(2):=0.0;
       end if;

       if NSOJS = 1 then
           TGT_POS_NED(3)(1):=COS(SOJ_ANGLE(1))*SQRT(TGT_RANGE(3)*
                              TGT_RANGE(3)-(-LAC_POS_NED(3)+
                              TGT_POS_NED(3)(3))*(-LAC_POS_NED(3)+
                              TGT_POS_NED(3)(3)));
           TGT_POS_NED(3)(2):=SIN(SOJ_ANGLE(1))*SQRT(TGT_RANGE(3)*
                              TGT_RANGE(3)-(-LAC_POS_NED(3)+
                              TGT_POS_NED(3)(3))*(-LAC_POS_NED(3)+
                              TGT_POS_NED(3)(3)));
           if NSOJS = 1 then
               TGT_POS_NED(4)(1):=0.0;
               TGT_POS_NED(4)(2):=0.0;
           end if;
       elsif NSOJS = 2 then
           TGT_POS_NED(3)(1):=COS(SOJ_ANGLE(1))*SQRT(TGT_RANGE(3)*
                              TGT_RANGE(3)-(-LAC_POS_NED(3)+
                              TGT_POS_NED(3)(3))*(-LAC_POS_NED(3)+
                              TGT_POS_NED(3)(3)));
           TGT_POS_NED(3)(2):=SIN(SOJ_ANGLE(1))*SQRT(TGT_RANGE(3)*
                              TGT_RANGE(3)-(-LAC_POS_NED(3)+
                              TGT_POS_NED(3)(3))*(-LAC_POS_NED(3)+
                              TGT_POS_NED(3)(3)));
           TGT_POS_NED(4)(1):=COS(SOJ_ANGLE(2))*SQRT(TGT_RANGE(4)*
                              TGT_RANGE(4)-(-LAC_POS_NED(3)+
                              TGT_POS_NED(4)(3))*(-LAC_POS_NED(3)+
                              TGT_POS_NED(4)(3)));
           TGT_POS_NED(4)(2):=SIN(SOJ_ANGLE(2))*SQRT(TGT_RANGE(4)*
                              TGT_RANGE(4)-(-LAC_POS_NED(3)+
                              TGT_POS_NED(4)(3))*(-LAC_POS_NED(3)+
                              TGT_POS_NED(4)(3)));
       else
           TGT_POS_NED(3)(1):=0.0;
           TGT_POS_NED(3)(2):=0.0;
           TGT_POS_NED(4)(1):=0.0;
           TGT_POS_NED(4)(2):=0.0;
       end if;

       if MANEUVER = TURN then
           FINAL_HEAD_ANGLE:=DEG_TO_RAD*REAL(integer((ASPECT(1)+
                   TURN_ANGLE)*RAD_TO_DEG) mod 360);
       elsif MANEUVER = WEAVE then
           OMEGA:=2.0*PI/WEAVE_PERIOD;
       end if;
   end INITIALIZE;
--------------------------------------------------------------------------
----------------------------------------------------------------------
   procedure TGT_DATA(TGT_VEL_OUT:  out VECTOR;
                      TGT_POS_OUT : out SUPER_VECTOR) is

   begin
       TGT_VEL_OUT:=TGT_VEL_NED;
       TGT_POS_OUT:=TGT_POS_NED;
   end TGT_DATA;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
   function TGT_POS return SUPER_VECTOR is

   begin
       return TGT_POS_NED;
```

```
    end TGT_POS;
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
    procedure COMPUTE is

        BUILDUP_FACTOR:  REAL;
        MANEUVER_START_VALUE: REAL;
        TEMP_GS:  REAL;
        TGT2_TEMP_ANGLE:      REAL;
    begin
        if not FSTART_MANEUVER then
            MISSILE.MANEUVER_VALUE(TURN_ON_PARAMETER,MANEUVER_START_VALUE);
        end if;

        if TURN_ON_PARAMETER /= FLIGHT_TIME and MANEUVER_START_VALUE <=
             TURN_ON_VALUE then
            if not FSTART_MANEUVER then
                START_TIME:=ENVIRONMENT.Time;
                FSTART_MANEUVER:=true;
            end if;
        elsif TURN_ON_PARAMETER = FLIGHT_TIME and MANEUVER_START_VALUE >=
             TURN_ON_VALUE then
            if not FSTART_MANEUVER then
                START_TIME:=ENVIRONMENT.Time;
                FSTART_MANEUVER:=true;
            end if;
        end if;

        if FSTART_MANEUVER and ENVIRONMENT.Time<=(START_TIME+
            BUILDUP_TIME) then
            if BUILDUP_TIME = 0.0 then
                TGT_HEAD_DOT:=SIGN(TURN_ANGLE)*G*SQRT(NO_OF_GS*NO_OF_GS-
                    1.0)/TGT_VEL;
                G_HORZ:=TAN(ACOS(1.0/NO_OF_GS));
            else
                BUILDUP_FACTOR:=1.0-EXP((ENVIRONMENT.Time-START_TIME)*
                    (-4.60517)/BUILDUP_TIME);
                if MANEUVER = TURN then
                     TEMP_GS:=NO_OF_GS*BUILDUP_FACTOR;
                    if TEMP_GS < 1.0 then
                        TEMP_GS:=1.0;
                    end if;
                    TGT_HEAD_DOT:=SIGN(TURN_ANGLE)*G*SQRT(TEMP_GS*TEMP_GS-
                        1.0)/TGT_VEL;
                elsif MANEUVER = WEAVE then
                    G_HORZ:=BUILDUP_FACTOR*TAN(ACOS(1.0/NO_OF_GS));
                end if;
            end if;
        end if;
        if MANEUVER = TURN and FSTART_MANEUVER then
            if (ASPECT(1)+TURN_ANGLE)<0.000 then
                if TGT_HEAD_ANGLE>ASPECT(1) and TGT_HEAD_ANGLE <=
                FINAL_HEAD_ANGLE then
                    TGT_HEAD_ANGLE:=FINAL_HEAD_ANGLE;
                else
                    TGT_HEAD_ANGLE:=ASPECT(1)+(ENVIRONMENT.Time-START_TIME)
                    *TGT_HEAD_DOT;
                end if;
            elsif  (ASPECT(1)+TURN_ANGLE) >=2.0 * PI then
                if TGT_HEAD_ANGLE<ASPECT(1) and TGT_HEAD_ANGLE >=
                FINAL_HEAD_ANGLE then
                    TGT_HEAD_ANGLE:=FINAL_HEAD_ANGLE;
                else
                    TGT_HEAD_ANGLE:=ASPECT(1)+(ENVIRONMENT.Time-START_TIME)
                    *TGT_HEAD_DOT;
                end if;
            elsif SIGN(TURN_ANGLE)<0.0 then
                 if FINAL_HEAD_ANGLE = 0.0 and TGT_HEAD_ANGLE >=6.20 then
                    TGT_HEAD_ANGLE:=FINAL_HEAD_ANGLE;
                elsif TGT_HEAD_ANGLE<=FINAL_HEAD_ANGLE then
                   TGT_HEAD_ANGLE:=FINAL_HEAD_ANGLE;
                else
                    TGT_HEAD_ANGLE:=ASPECT(1)+(ENVIRONMENT.Time-START_TIME)*
                    TGT_HEAD_DOT;
                end if;
            else
                if TGT_HEAD_ANGLE>=FINAL_HEAD_ANGLE then
                   TGT_HEAD_ANGLE:=FINAL_HEAD_ANGLE;
                else
                    TGT_HEAD_ANGLE:=ASPECT(1)+(ENVIRONMENT.Time-START_TIME)*
                    TGT_HEAD_DOT;
               end if;
            end if;
        elsif MANEUVER = WEAVE and FSTART_MANEUVER then
            TGT_HEAD_ANGLE:=ASPECT(1)-G*G_HORZ*COS(OMEGA*
               (ENVIRONMENT.Time-START_TIME))/TGT_VEL/OMEGA;
        end if;
        if TGT_HEAD_ANGLE < 0.0 or TGT_HEAD_ANGLE >= 2.0*PI then
            TGT_HEAD_ANGLE:=DEG_TO_RAD*REAL(integer(TGT_HEAD_ANGLE
                  *RAD_TO_DEG) mod 360);
        end if;
        TGT_VEL_NED(1):=TGT_VEL*COS(TGT_HEAD_ANGLE);
        TGT_VEL_NED(2):=TGT_VEL*SIN(TGT_HEAD_ANGLE);
        TGT_VEL:=MAGNITUDE(TGT_VEL_NED);
        TGT_MACH:=TGT_VEL/ENVIRONMENT.SPEED_OF_SOUND(-TGT_POS_NED(1)(3));
        if NTGTS = 2 then
             if MANEUVER = TURN then
                 TGT2_TEMP_ANGLE:=TGT_HEAD_ANGLE+TGT2_ANGLE-PI;
                 TGT_POS_NED(2)(1):=TGT_POS_NED(1)(1)+SIN(TGT2_TEMP_ANGLE)*
                                    TGT_RANGE(2);
                 TGT_POS_NED(2)(2):=TGT_POS_NED(1)(2)-COS(TGT2_TEMP_ANGLE)*
                                    TGT_RANGE(2);
             else
                 TGT_POS_NED(2)(1):=TGT_POS_NED(1)(1)+SIN(TGT2_ANGLE)*
                                    TGT_RANGE(2);
                 TGT_POS_NED(2)(2):=TGT_POS_NED(1)(2)-COS(TGT2_ANGLE)*
                                    TGT_RANGE(2);
             end if;
        end if;
    end COMPUTE;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function TGT_ASPECT(RANGE_VEC:  in VECTOR) return REAL is

    ASPECT : REAL;
    begin
        if RANGE_VEC(2) /= 0.0 then
            ASPECT:=DEG_TO_RAD*REAL(INTEGER(RAD_TO_DEG*(TGT_HEAD_ANGLE-
                       ATAN(RANGE_VEC(2)/RANGE_VEC(1))))mod 360);
        else
            ASPECT:=TGT_HEAD_ANGLE;
        end if;
        return ASPECT;
    end TGT_ASPECT;

end TARGETS;

-----------------------------------------------------------------------
-- Environment Package
--
-- This package contains procedures to set atmospheric conditions
-- according to altitude.  It uses standard temperature and pressure
-- for each altitude zone.
--
-----------------------------------------------------------------------

with MATH; use MATH;
```

```
package ENVIRONMENT is

    procedure SET_TIME(
        NEW_TIME : in REAL);

    function Time
    return REAL;

    function AIR_DENSITY(ALTITUDE : in REAL)
    return REAL;

    function SPEED_OF_SOUND(ALTITUDE : in REAL)
    return REAL;

end ENVIRONMENT;

--------------------------------------------------------------------------------
-- Environment Package
--
-- This package contains procedures to set the time, return the time,
-- compute position and gravity vectors due to the earth's curvature,
-- and set atmospheric conditions according to altitude.
--
--------------------------------------------------------------------------------

with MATH; use MATH;
with REAL_MATRIX; use REAL_MATRIX;

package body ENVIRONMENT is

    SYSTEM_TIME : REAL := 0.0;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    procedure SET_TIME(
        NEW_TIME : in REAL) is
    begin
        SYSTEM_TIME := NEW_TIME;
    end SET_TIME;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function TIME return REAL is
    begin
        return SYSTEM_TIME;
    end Time;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function AIR_DENSITY(
        ALTITUDE       : in REAL)  return REAL is

        T0 : constant := 518.69;
        RHO0 : constant := 2.3769E-3;

        H1 : constant := 36500.0;
        T1 : constant := 389.99;
        RHO1 : constant := 6.9443E-4;

        H2 : constant := 82000.0;
        T2 : constant := 389.99;
        RHO2 : constant := 7.8931E-5;

        H3 : constant := 156000.0;
        T3 : constant := 508.79;

        A1 : constant := (T1 - T0) / H1;
        A3 : constant := (T3 - T2) / (H3 - H2);

        INV_T0 : constant := 1.0 / T0;
        EXP_0 : constant := -(G / (A1 * R) +1.0);
        EXP_COEF_1 : constant := -G / (R * T1);

        INV_T2 : constant := 1.0 / T2;
        EXP_2 : constant := -(G / (A3 *R) + 1.0);

        RHO : REAL;
        TEMP : REAL;

    begin
        if ALTITUDE < H1 then
            TEMP := T0 + ALTITUDE * A1;
            RHO := RHO0 * REAL(TEMP * INV_T0) ** REAL(EXP_0);
        elsif (ALTITUDE >= H1) and (ALTITUDE < H2) then
            RHO := RHO1 * EXP(EXP_COEF_1 * (ALTITUDE - H1));
        elsif ALTITUDE >= H2 then
            TEMP := T2 + (ALTITUDE - H2) * A3;
            RHO := RHO2 * REAL(TEMP * INV_T2)**REAL(EXP_2);
        end if;
        return RHO;
    end AIR_DENSITY;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    function SPEED_OF_SOUND(
        ALTITUDE        : in REAL)  return REAL is

        T0 : constant := 518.69;

        H1 : constant := 36500.0;
        T1 : constant := 389.99;

        H2 : constant := 82000.0;
        T2 : constant := 389.99;

        H3 : constant := 156000.0;
        T3 : constant := 508.79;

        A1 : constant := (T1 - T0) / H1;
        A3 : constant := (T3 - T2) / (H3 - H2);

        TEMP : REAL;
        SOUND_SPEED : REAL;

    begin
        if ALTITUDE < H1 then
            TEMP := T0 + ALTITUDE * A1;
            SOUND_SPEED := SQRT(GAMMA * R * TEMP);
        elsif (ALTITUDE >= H1) and (ALTITUDE < H2) then
            TEMP := T1;
            SOUND_SPEED := SQRT(GAMMA * R * TEMP);
        elsif ALTITUDE >= H2 then
            TEMP := T2 + (ALTITUDE - H2) * A3;
            SOUND_SPEED := SQRT(GAMMA * R * TEMP);
        end if;
        return SOUND_SPEED;
    end SPEED_OF_SOUND;

end ENVIRONMENT;
```

LIST OF REFERENCES

1. Pacific Missile Test Center, *Test and Evaluation of an Air-to-Air RF Guided Missile*, by E.J. Eichblatt, 1 July 1981

2. Pressman, R.S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill Book Company, 1987

3. Myers,G., *Composite Structured Design*, Van Nordstrand Inc., 1978

4. Cox,B.J., *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley Publishing Co., 1987

5. Booch, G., *Software Engineering with Ada*, The Benjamin/Cummings Publishing Company, Inc., 1986

6. ANSI/MIL-STD-1815A, *Ada Programming Language*, 22 January 1983

7. Cohen,N.H., *Ada as a Second Language*, McGraw-Hill, Inc., 1986

8. Abbot, R.,"Program Design by Informal English Description," *Communications of the ACM*, 1983

9. Anderson, J.D., *Introduction to Flight*, McGraw-Hill Inc., 1987

10. Smith, J.M., *Mathematical Modeling and Digital Simulation for Engineers and Scientists*, John Wiley & Sons, Inc., 1987

11. Gerald,C.F., and Wheatley,P.O., *Applied Numerical Analysis*, Addison-Wesley Publishing Co., 1984

12. Howe,R.M., *Dynamics of Real-Time Simulation*, Applied Dynamics International, 1987

13. Meridian Software Systems Inc., *Meridian AdaVantage DOS Environment Library User's Guide*, Meridian Inc.

14. Meridian Software Systems Inc., *Meridian AdaVantage Compiler User's Guide For PC-DOS Self-Targeted Configurations*, Meridian Inc.

15. Young,M.J., *MS DOS Advanced Programming*, SYBEX Inc.,1988

16. Hanna, O.T., *The Improved Euler Extrapolation Integration Algorithm for Use in Digital Simulation*, Integrated System Analysts Inc.,1990

17. Konvisser,M.W., *Elementary Linear Algebra with Applications*, Prindle, Weber, and Schmidt, 1981

18. Air Force Flight Dynamics Laboratory Report RTD-TDR-64-1, *Six-Degree-of-Freedom Flight-Path Generalized Computer Program*, by R.C. Brown, R.V. Brulle, A.E. Combs and G.D. Giffin, October 1964

19. Criel, H.E. and Murtaugh, S.A.,"Fundamentals of Proportional Navigation," *IEEE Spectrum,* December 1964

20. Friedman,D.R., *Principles of Naval Weapons Systems*, Naval Institute Press, 1986

21. Stimson,.G.W., *Introduction to Airborne Radar*, Hughes Aircraft Company, 1983

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Professor Y. Lee, Code CS/Le Naval Postgraduate School Monterey, CA 93943-50000	2
4.	Mr. John V. Waite, Code 1051 Pacific Missile Test Center Point Mugu, CA 93042	2

Thesis
W21775 Waite
c.1 An Ada object oriented
missile flight simulation.